SESSION 5

iOS DEVELOPMENT

BUILDING FOR EVERY DEVICE

LAYOUT

BUILDING FOR EVERY DEVICE

- So far this quarter, we've been ignoring a few important facts:
    - 1. iPhones come in many different sizes.
    - 2. They can be rotated from portrait to landscape.
    - 3. iPads exist!

BUILDING FOR EVERY DEVICE

- When you create an iOS app, you need to decide:
  - Does it support iPhone, iPad or both?
  - Does it support landscape, portrait or both?
- The layout of your app will need to take into account these choices.

BUILDING FOR EVERY DEVICE

- Even if you only support iPhones in portrait mode, you'll still need to build for a wide range of screen sizes.

iPhone SE

- Silver, space gray, gold, and rose gold
- 4-inch Retina display
- Single 12MP camera (Wide) with HDR and 4K video at 30 fps

iPhone 11 Pro Max

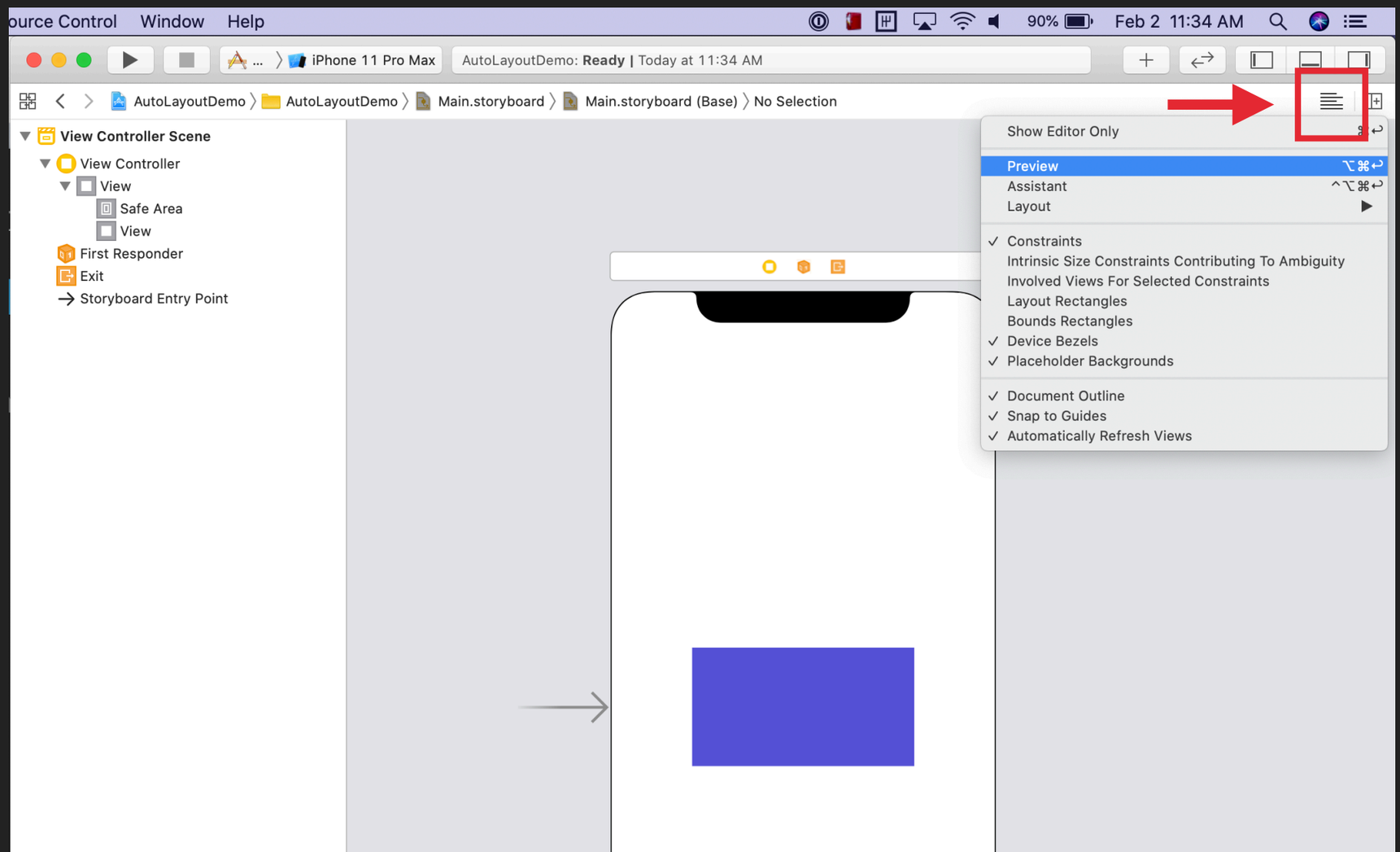
- Silver, space gray, gold, and midnight green
- 6.5-inch Super Retina XDR display with HDR and True Tone[1]

BUILDING FOR EVERY DEVICE

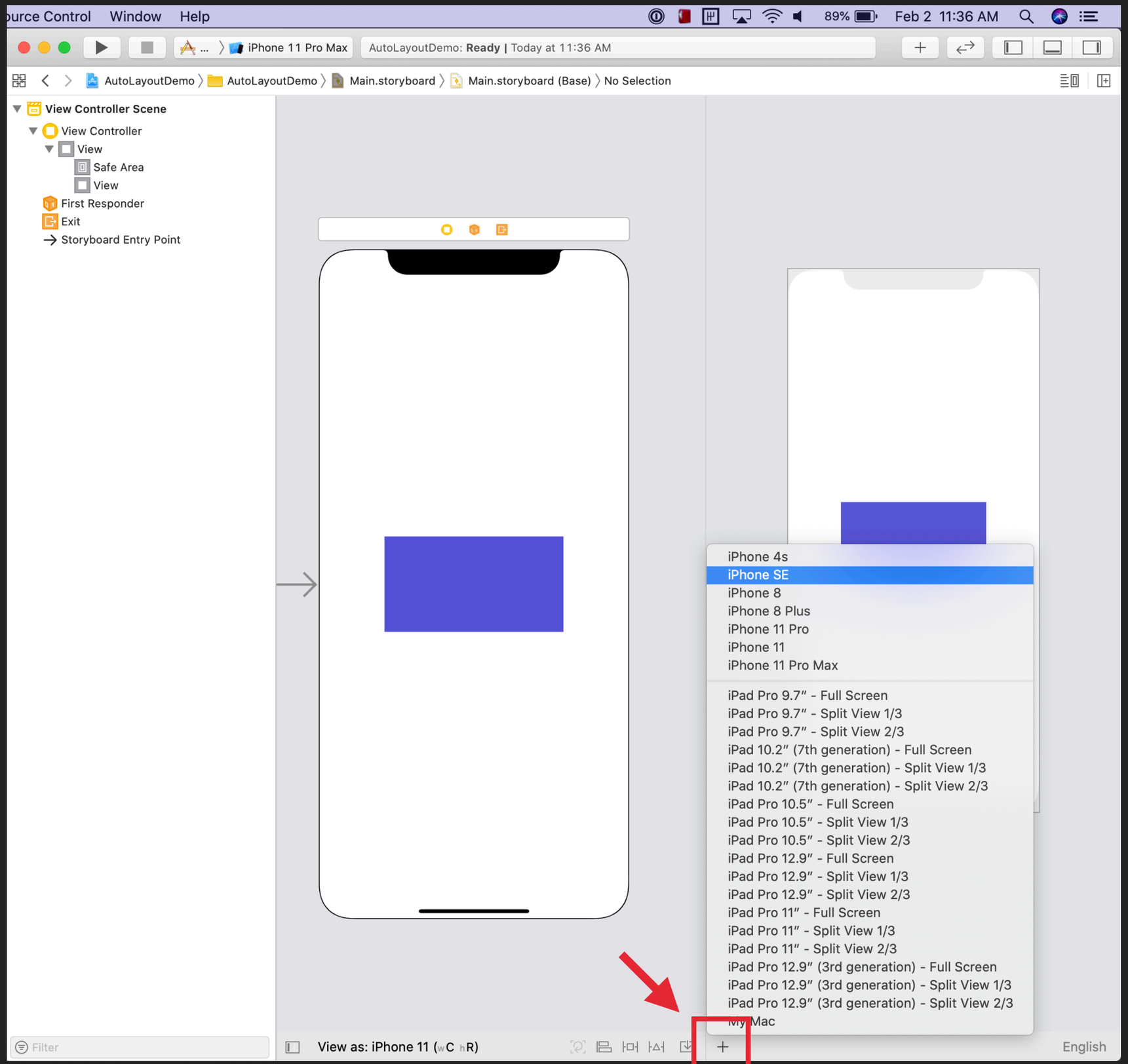
- The size of the root view in a view hierarchy will change depending on device size and orientation.
- Question: What happens to a subview's frame when you modify the frame of its superview?

BUILDING FOR EVERY DEVICE

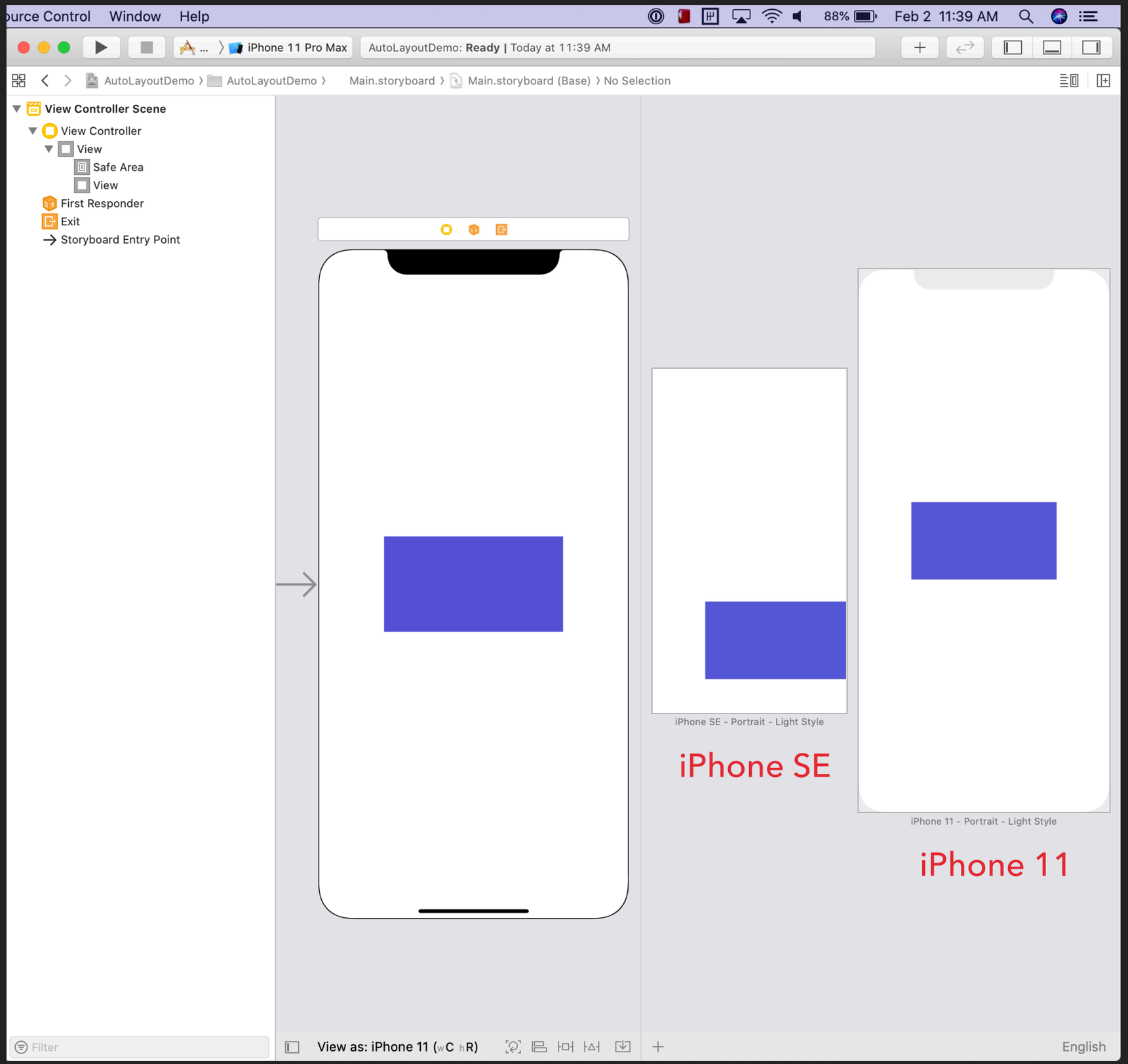
- An easy way to see this is with the Preview window in Xcode.



- Click on the + button to add previews of different devices



- The subview is centered on iPhone 11, but off-center on iPhone SE



ADJUSTING LAYOUTS

- Suppose we want a subview to remain centered, no matter what size its superview is.
- There are 3 ways to achieve this:
  - Auto Layout
  - Autoresizing
  - Manual Layout

AUTO LAYOUT

- Auto Layout determines the layout of views according to the constraints you set
- A constraint is defined with an instance of `NSLayoutConstraint`

AUTO LAYOUT

- Each constraint describes either:
    - The relationship between an attribute of one view and an attribute of another view

    -OR-

    - The absolute width or height of a single view

AUTO LAYOUT

- When describing the relationship between two views, the possible attributes are:
  - `.width`, `.height`
  - `.top`, `.bottom`
  - `.left`, `.right`, `.leading`, `.trailing`
  - `.centerX`, `.centerY`
  - `.firstBaseline`, `.lastBaseline`

AUTO LAYOUT

- You can specify a multiplier and a constant when you describe the relationship between two attributes

AUTO LAYOUT

- In addition to specifying that two attributes are equal, you can specify other relations:
  - `.equal`
  - `.greaterThanOrEqual`
  - `.lessThanOrEqual`

view1.width >= (view2.width * 2) + 50

Relation

AUTO LAYOUT

- Initializing an `NSLayoutConstraint`

```
NSLayoutConstraint(item: view1,
attribute: .width,
relatedBy: .equal,
toItem: view2,
attribute: .width,
multiplier: 2,
constant: 50)
```

First view and its attribute

AUTO LAYOUT

- Initializing an `NSLayoutConstraint`

```
NSLayoutConstraint(item: view1,
                   attribute: .width,
                   relatedBy: .equal,
                   toItem: view2,
                   attribute: .width,
                   multiplier: 2,
                   constant: 50)
```

Relation → `relatedBy: .equal,`

AUTO LAYOUT

- Initializing an `NSLayoutConstraint`

```
NSLayoutConstraint(item: view1,
                   attribute: .width,
                   relatedBy: .equal,
                   toItem: view2,
                   attribute: .width,
                   multiplier: 2,
                   constant: 50)
```

Second view and its attribute →

AUTO LAYOUT

- Initializing an `NSLayoutConstraint`

```
NSLayoutConstraint(item: view1,
                   attribute: .width,
                   relatedBy: .equal,
                   toItem: view2,
                   attribute: .width,
                   multiplier: 2,
                   constant: 50)
```

Multiplier and constant →

AUTO LAYOUT

- Initializing an `NSLayoutConstraint` for an absolute width or height value

Pass `nil` for the second item and `.notAnAttribute` for the second attribute →

```
NSLayoutConstraint(item: subview2 as Any,
                   attribute: .width,
                   relatedBy: .equal,
                   toItem: nil,
                   attribute: .notAnAttribute,
                   multiplier: 1,
                   constant: 100)
```

AUTO LAYOUT

- Constraints need to activated in order for them to take effect.
- The easiest way to do this is by passing an array of constraints into `NSLayoutConstraint.activate`
  - Static method
  - Takes an array of constraints as input

AUTO LAYOUT

- You can activate and deactivate constraints as needed throughout the lifecycle of your application
  - `NSLayoutConstraint.activate`
  - `NSLayoutConstraint.deactivate`

AUTO LAYOUT

- The code to create constraints programmatically can be very long
- There are a few other options that require less code

```
NSLayoutConstraint(item: subview1 as Any,
                   attribute: .top,
                   relatedBy: .equal,
                   toItem: view,
                   attribute: .top,
                   multiplier: 1,
                   constant: 0),

NSLayoutConstraint(item: subview1 as Any,
                   attribute: .leading,
                   relatedBy: .equal,
                   toItem: view,
                   attribute: .leading,
                   multiplier: 1,
                   constant: 0),

NSLayoutConstraint(item: subview1 as Any,
                   attribute: .trailing,
                   relatedBy: .equal,
                   toItem: view,
                   attribute: .trailing,
                   multiplier: 1,
                   constant: 0),

NSLayoutConstraint(item: subview1 as Any,
                   attribute: .height,
                   relatedBy: .equal,
                   toItem: nil,
                   attribute: .notAnAttribute,
                   multiplier: 1,
                   constant: 100),

NSLayoutConstraint(item: subview2 as Any,
                   attribute: .width,
```

AUTO LAYOUT

- Option 1: Anchor Notation

```
view1.widthAnchor.constraint(equalTo: view2.widthAnchor,
                             multiplier: 2,
                             constant: 50)
```

view1.width = (view2.width * 2) + 50

AUTO LAYOUT

- Option 2: Visual Format Language

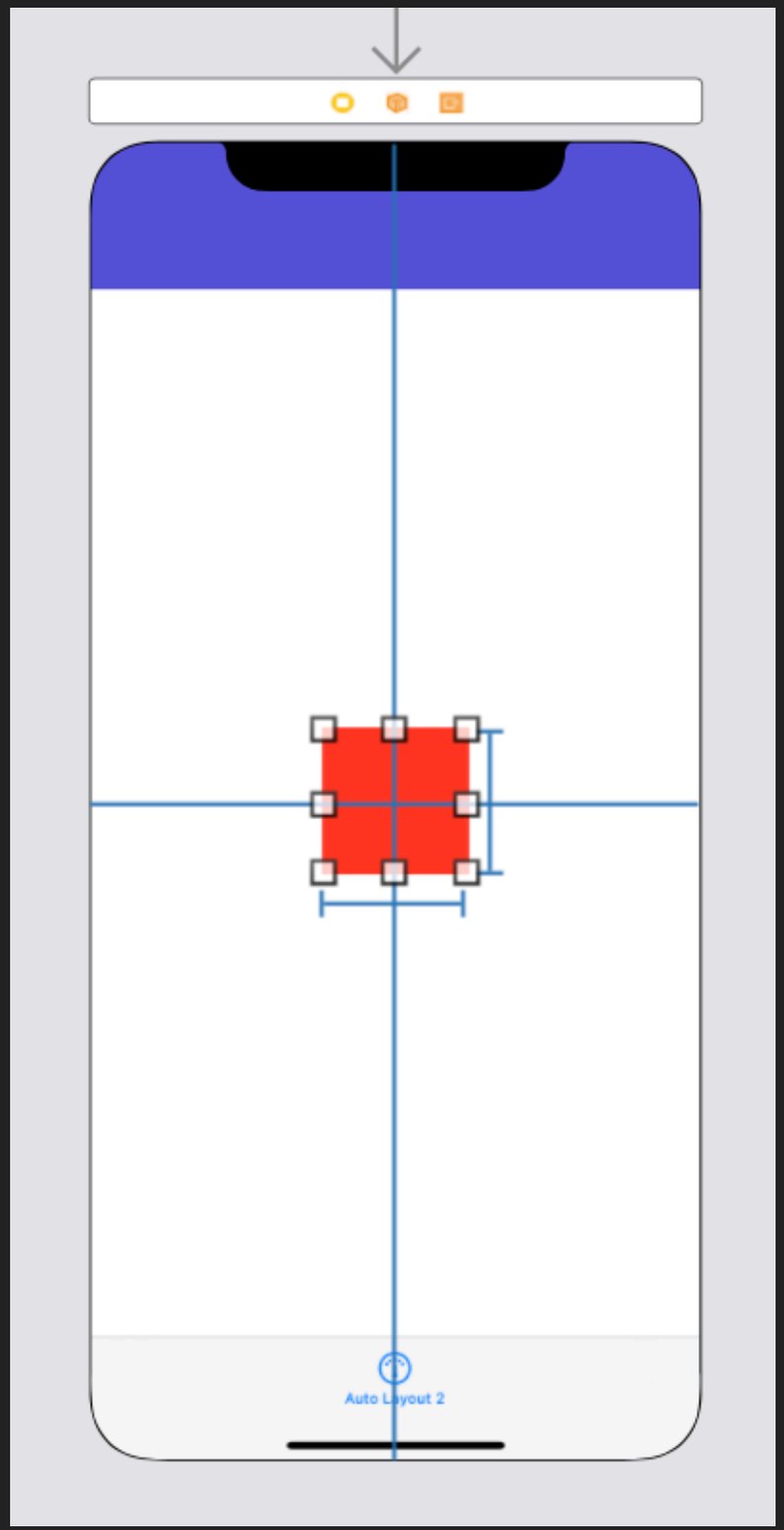
```
NSLayoutConstraint.constraints(
        withVisualFormat: "V:|[subview1(100)]",
        metrics: nil,
        views: views)
```

Dictionary of strings used to represent each view

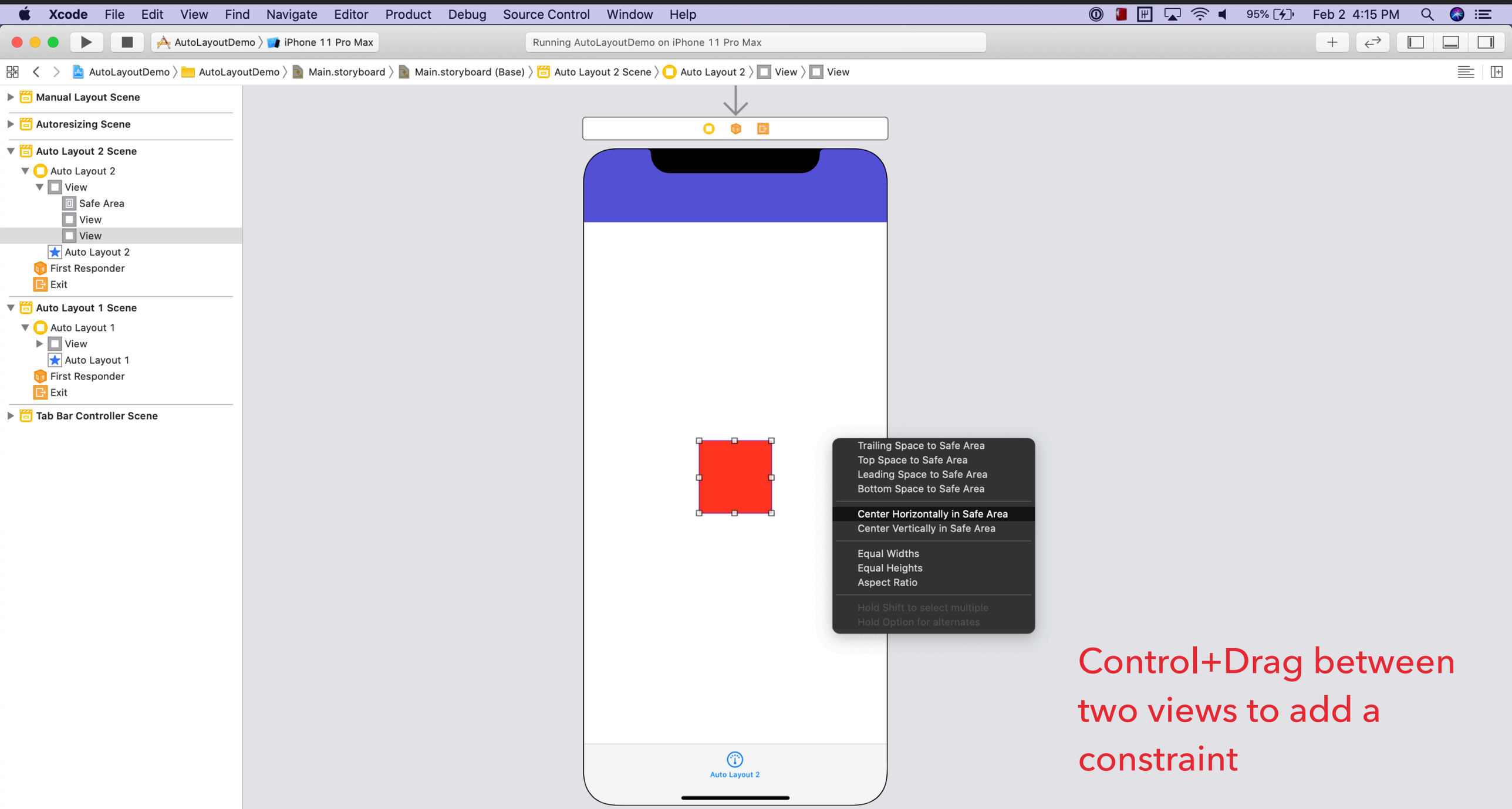
Pin subview1 to the top of its superview and give it a height of 100

AUTO LAYOUT

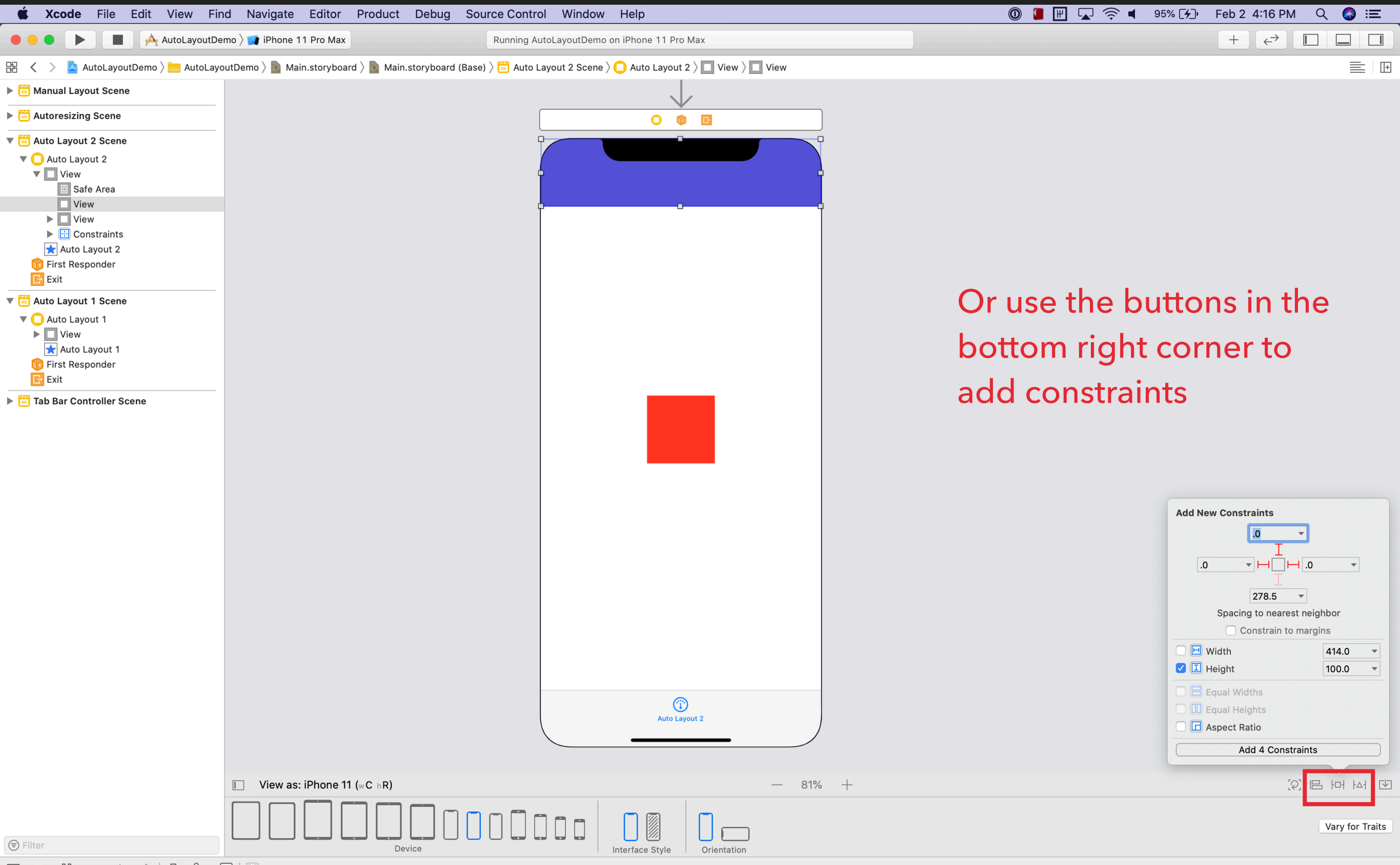
- Option 3: Interface Builder

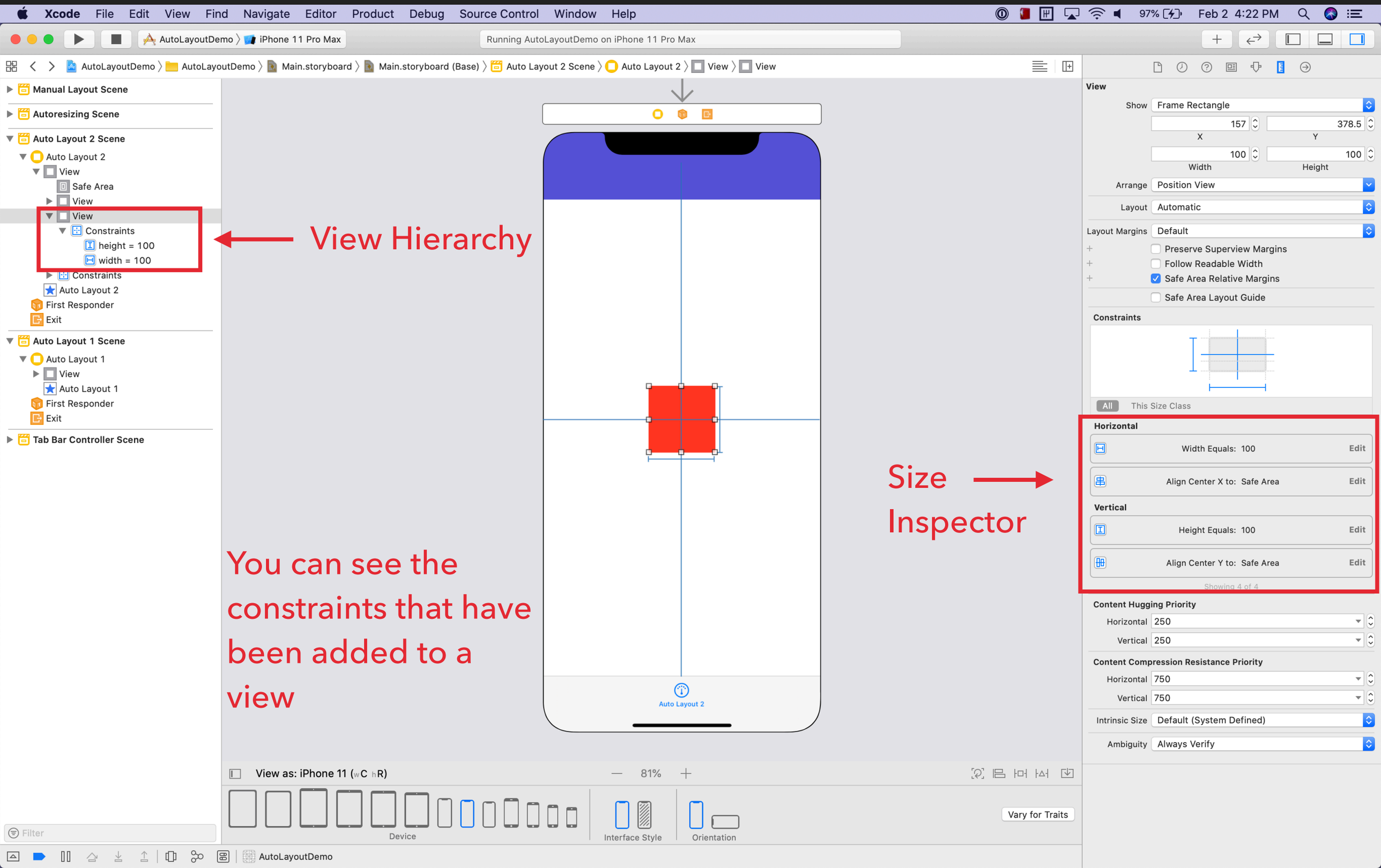


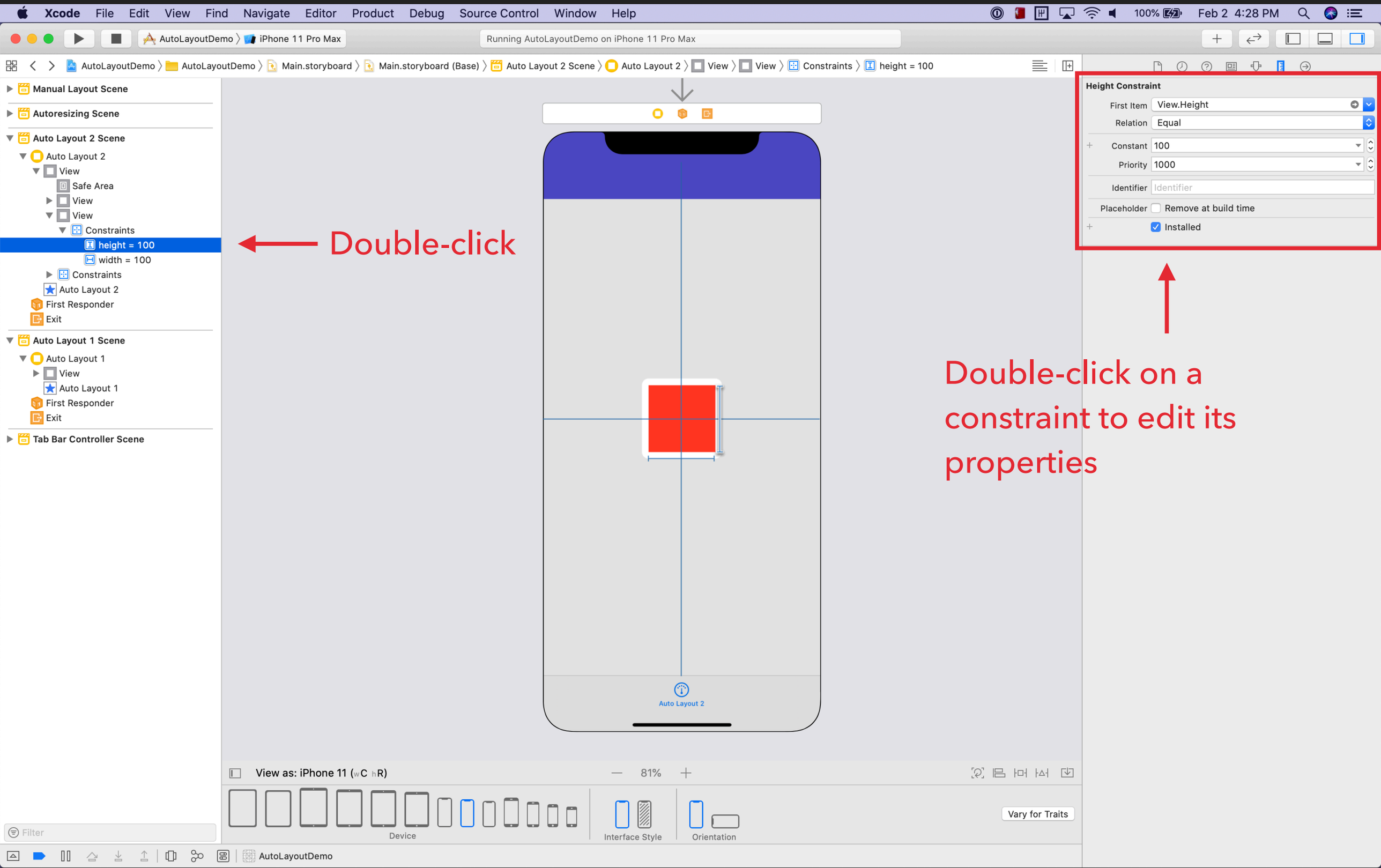
AUTO LAYOUT IN INTERFACE BUILDER



Control+Drag between two views to add a constraint

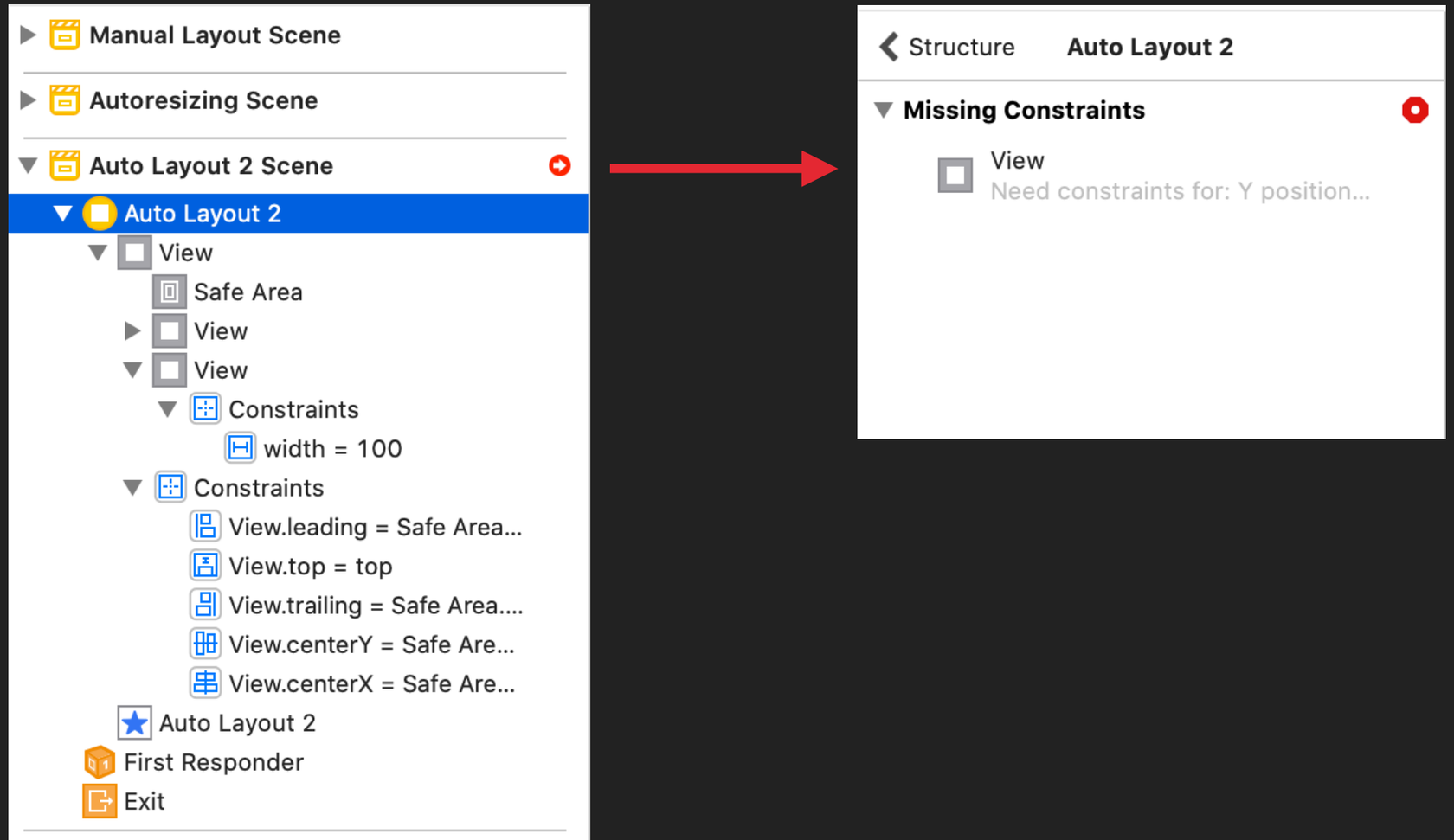

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
95% Feb 2 4:16 PM
AutoLayoutDemo iPhone 11 Pro Max
Running AutoLayoutDemo on iPhone 11 Pro Max
AutoLayoutDemo AutoLayoutDemo Main.storyboard Main.storyboard (Base) Auto Layout 2 Scene Auto Layout 2 View View
Manual Layout Scene
Autoresizing Scene
Auto Layout 2 Scene
Auto Layout 2
View
Safe Area
View
View
Constraints
Auto Layout 2
First Responder
Exit
Auto Layout 1 Scene
Auto Layout 1
View
Auto Layout 1
First Responder
Exit
Tab Bar Controller Scene
Auto Layout 2
Or use the buttons in the bottom right corner to add constraints
Add New Constraints
.0
.0
.0
278.5
Spacing to nearest neighbor
Constrain to margins
Width 414.0
Height 100.0
Equal Widths
Equal Heights
Aspect Ratio
Add 4 Constraints
View as: iPhone 11 (wC hR)
81%
Device
Interface Style
Orientation
Vary for Traits
Filter
AutoLayoutDemo


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
97% Feb 2 4:22 PM
AutoLayoutDemo iPhone 11 Pro Max
Running AutoLayoutDemo on iPhone 11 Pro Max
AutoLayoutDemo AutoLayoutDemo Main.storyboard Main.storyboard (Base) Auto Layout 2 Scene Auto Layout 2 View View
Manual Layout Scene
Autoresizing Scene
Auto Layout 2 Scene
Auto Layout 2
View
Safe Area
View
View
Constraints
height = 100
width = 100
Constraints
Auto Layout 2
First Responder
Exit
Auto Layout 1 Scene
Auto Layout 1
View
Auto Layout 1
First Responder
Exit
Tab Bar Controller Scene
View Hierarchy
You can see the constraints that have been added to a view
Auto Layout 2
View as: iPhone 11 (wC hR)
81%
Device
Interface Style
Orientation
Vary for Traits
AutoLayoutDemo
View
Show Frame Rectangle
X 157
Y 378.5
Width 100
Height 100
Arrange Position View
Layout Automatic
Layout Margins Default
Preserve Superview Margins
Follow Readable Width
Safe Area Relative Margins
Safe Area Layout Guide
Constraints
All This Size Class
Horizontal
Width Equals: 100 Edit
Align Center X to: Safe Area Edit
Vertical
Height Equals: 100 Edit
Align Center Y to: Safe Area Edit
Showing 4 of 4
Size Inspector
Content Hugging Priority
Horizontal 250
Vertical 250
Content Compression Resistance Priority
Horizontal 750
Vertical 750
Intrinsic Size Default (System Defined)
Ambiguity Always Verify


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
100% Feb 2 4:28 PM
AutoLayoutDemo iPhone 11 Pro Max
Running AutoLayoutDemo on iPhone 11 Pro Max
AutoLayoutDemo AutoLayoutDemo Main.storyboard Main.storyboard (Base) Auto Layout 2 Scene Auto Layout 2 View View Constraints height = 100
Manual Layout Scene
Autoresizing Scene
Auto Layout 2 Scene
Auto Layout 2
View
Safe Area
View
View
Constraints
height = 100
width = 100
Constraints
Auto Layout 2
First Responder
Exit
Auto Layout 1 Scene
Auto Layout 1
View
Auto Layout 1
First Responder
Exit
Tab Bar Controller Scene
Filter
Double-click
Auto Layout 2
Height Constraint
First Item View.Height
Relation Equal
Constant 100
Priority 1000
Identifier Identifier
Placeholder Remove at build time
Installed
Double-click on a constraint to edit its properties
View as: iPhone 11 (wC hR)
81%
Device
Interface Style
Orientation
Vary for Traits
AutoLayoutDemo

AUTO LAYOUT IN INTERFACE BUILDER

- One big advantage of adding constraints in Interface Builder is that it will tell you when there's a problem



AUTO LAYOUT IN INTERFACE BUILDER

- 3 main problems you'll see:
  - Conflicting constraints
  - Missing constraints
  - Misplaced views

CONFLICTING CONSTRAINTS

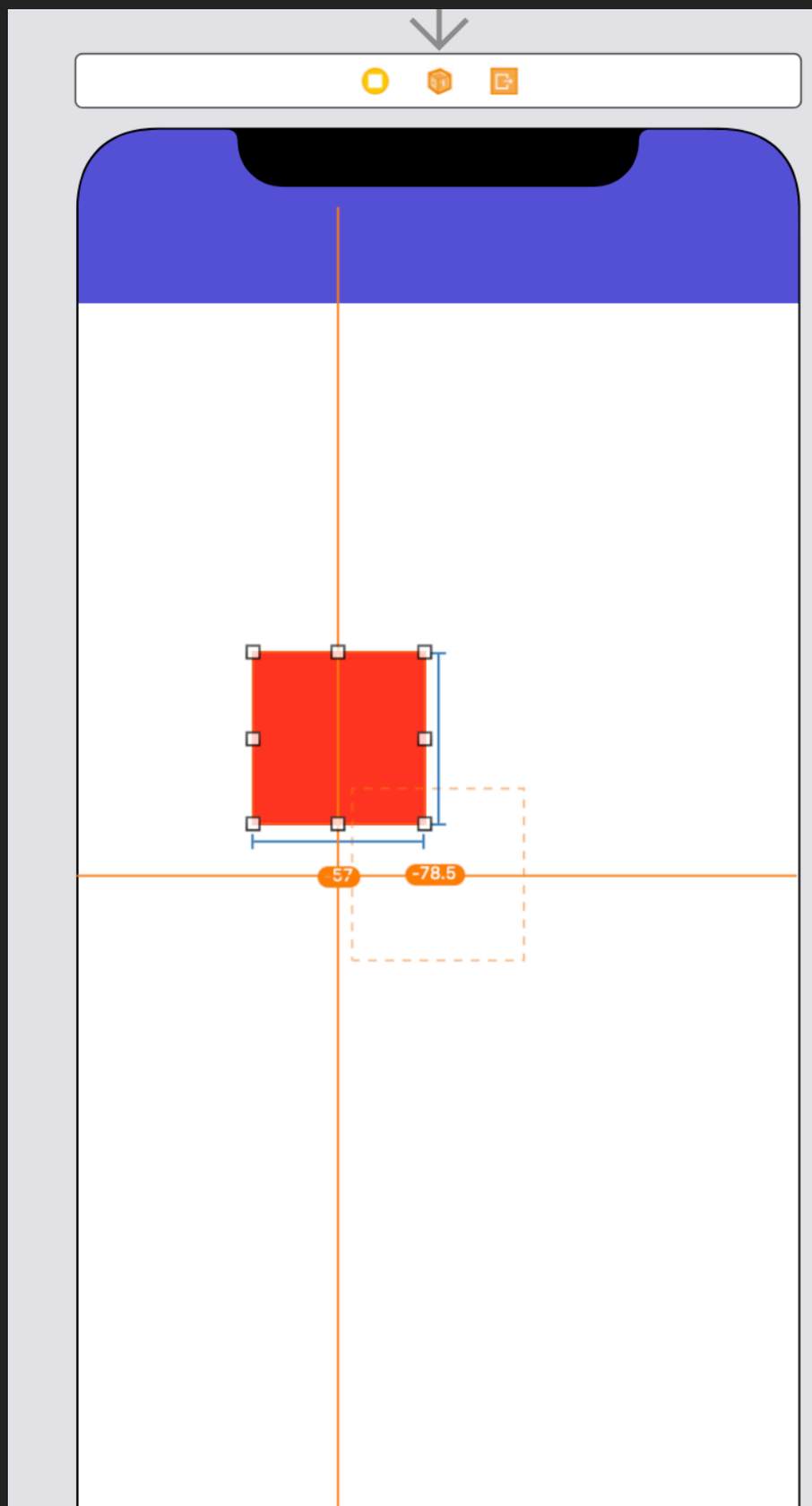
- A conflict between constraints means the constraints on a view cannot all be enforced simultaneously
    - Example:
        - 1: Center horizontally in superview
        - 2: Align leading edge of view with leading edge of superview
        - 3: Width constraint of 50

MISSING CONSTRAINTS

- A missing constraint occurs when there are not enough constraints to determine the placement of the view
    - Example:
        - 1: Center horizontally in superview
        - 2: Center vertically in superview
        - 3: Height constraint of 20

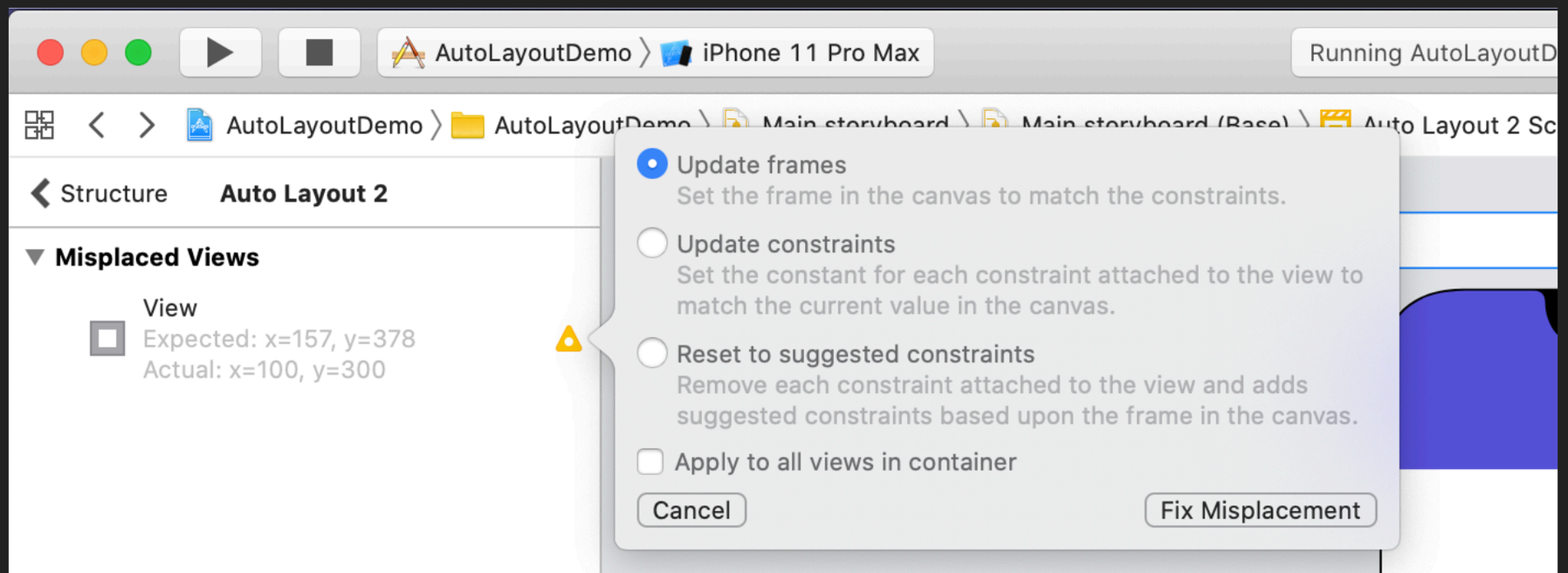
MISPLACED VIEWS

- A view is misplaced if its position on storyboard does not match the position determined by its constraints



MISPLACED VIEWS

- Xcode will offer a few ways of fixing this:
  - Update frames
  - Update constraints
  - Reset to suggested constraints



BUILDING FOR EVERY DEVICE

SIZE CLASSES

WHY SIZE CLASSES?

- The number of available iOS devices has grown dramatically since the first iPhone was introduced

Device
12.9" iPad Pro
11" iPad Pro
10.5" iPad Pro
9.7" iPad
7.9" iPad mini 4
iPhone Xs Max
iPhone Xs
iPhone Xʀ
iPhone X
iPhone 8 Plus
iPhone 8
iPhone 7 Plus
iPhone 7
iPhone 6s Plus
iPhone 6s
iPhone SE

WHY SIZE CLASSES?

- Writing code for every possible device and orientation would be time-consuming and error-prone

Device
12.9" iPad Pro
11" iPad Pro
10.5" iPad Pro
9.7" iPad
7.9" iPad mini 4
iPhone Xs Max
iPhone Xs
iPhone Xʀ
iPhone X
iPhone 8 Plus
iPhone 8
iPhone 7 Plus
iPhone 7
iPhone 6s Plus
iPhone 6s
iPhone SE

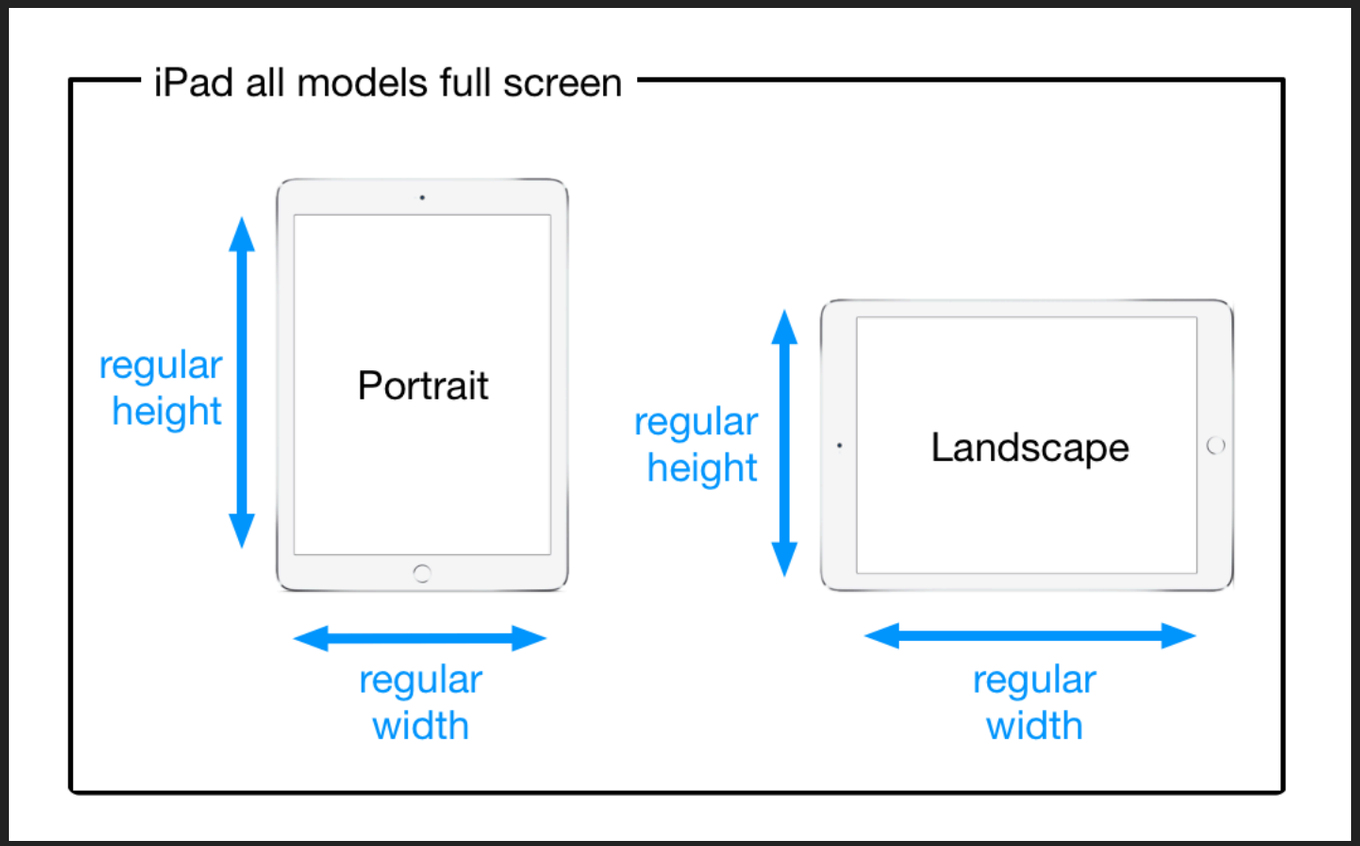
WHY SIZE CLASSES?

- Instead, Apple introduced the notion of *size classes*
  - Regular
  - Compact

Device
12.9" iPad Pro
11" iPad Pro
10.5" iPad Pro
9.7" iPad
7.9" iPad mini 4
iPhone Xs Max
iPhone Xs
iPhone Xʀ
iPhone X
iPhone 8 Plus
iPhone 8
iPhone 7 Plus
iPhone 7
iPhone 6s Plus
iPhone 6s
iPhone SE

THINGS STARTED SIMPLE...

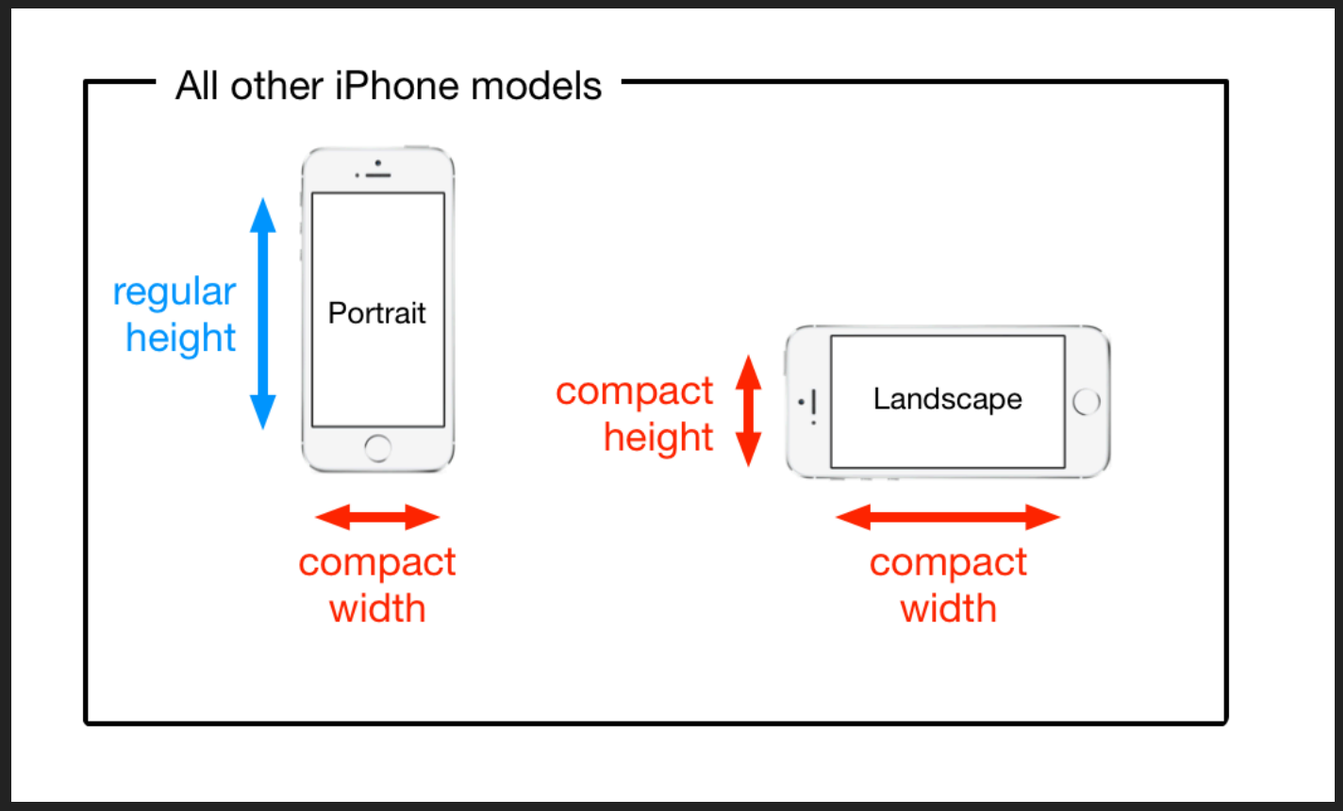
- iPads were always regular height and regular width



https://useyourloaf.com/blog/size-classes/

THINGS STARTED SIMPLE...

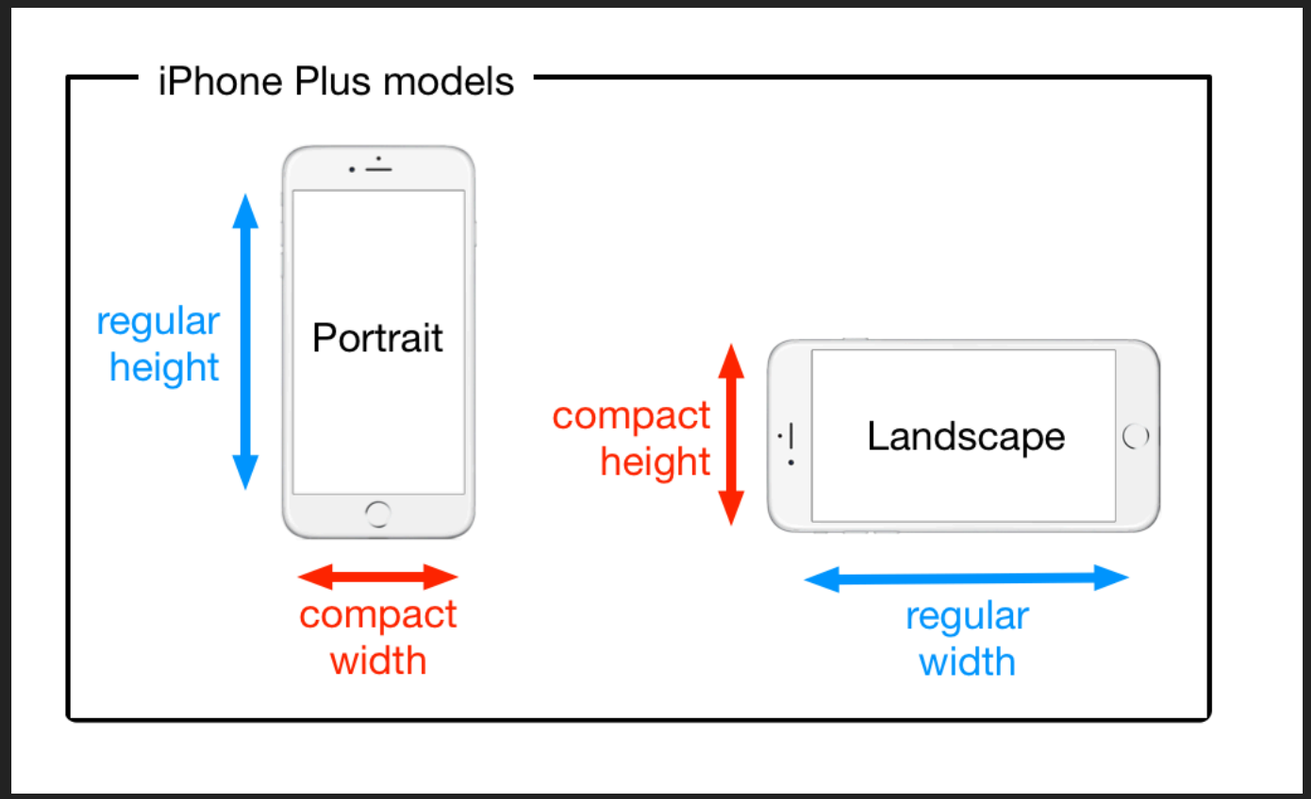
- iPhones were always compact width with regular height (portrait) or compact height (landscape)



https://useyourloaf.com/blog/size-classes/

THEN PHONES GOT BIGGER

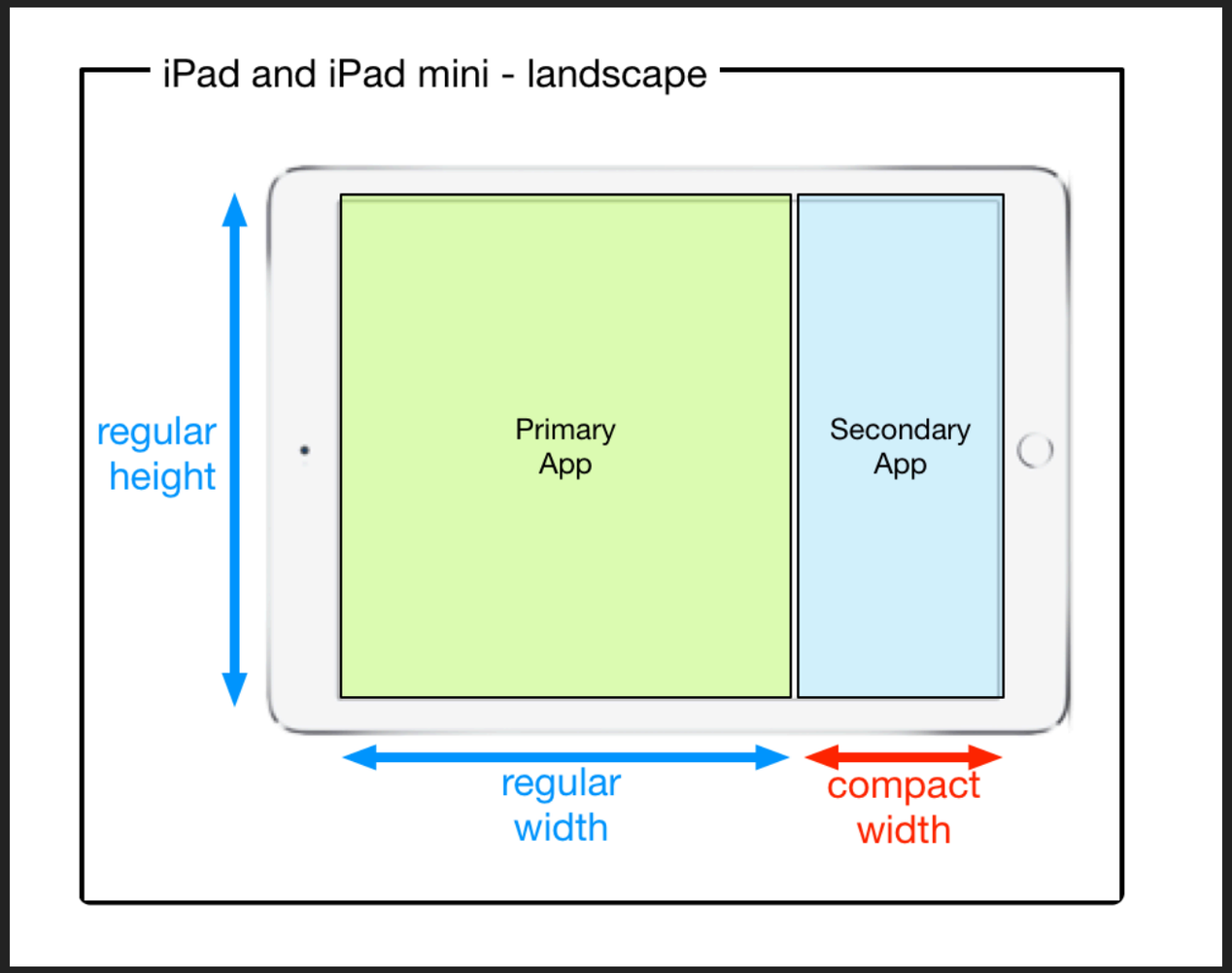
- iPhone Plus models have regular width in landscape



https://useyourloaf.com/blog/size-classes/

iPAD APPS STARTED SHARING THE SCREEN

- An iPad app in multitasking mode can have compact width



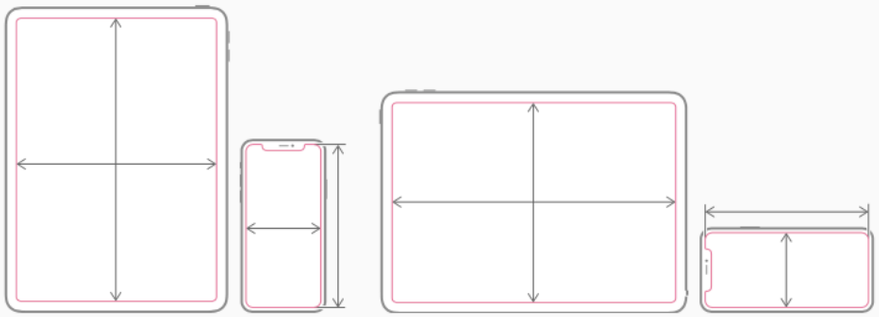
https://useyourloaf.com/blog/size-classes/

NOW WE HAVE THIS

https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/adaptivity-and-layout/

Device Size Classes

Different size class combinations apply to the full-screen experience on different devices, based on screen size.



Device	Portrait orientation	Landscape orientation
12.9" iPad Pro	Regular width, regular height	Regular width, regular height
11" iPad Pro	Regular width, regular height	Regular width, regular height
10.5" iPad Pro	Regular width, regular height	Regular width, regular height
9.7" iPad	Regular width, regular height	Regular width, regular height
7.9" iPad mini 4	Regular width, regular height	Regular width, regular height
iPhone Xs Max	Compact width, regular height	Regular width, compact height
iPhone Xs	Compact width, regular height	Compact width, compact height
iPhone Xʀ	Compact width, regular height	Regular width, compact height
iPhone X	Compact width, regular height	Compact width, compact height
iPhone 8 Plus	Compact width, regular	Regular width, compact

Multitasking Size Classes

On iPad, size classes also apply when your app runs in a multitasking configuration.

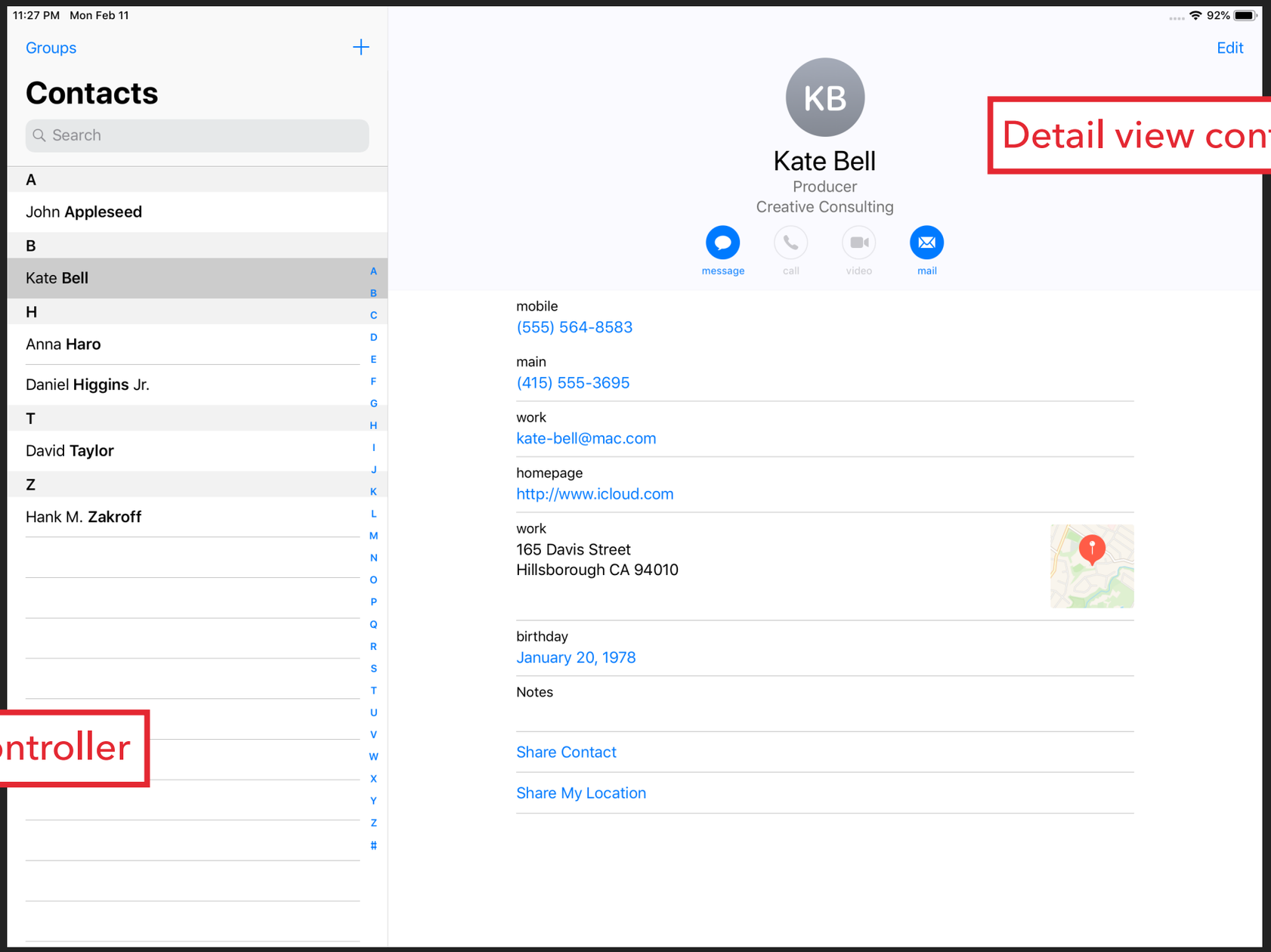
2/3 split view

1/2 split view

1/3 split view

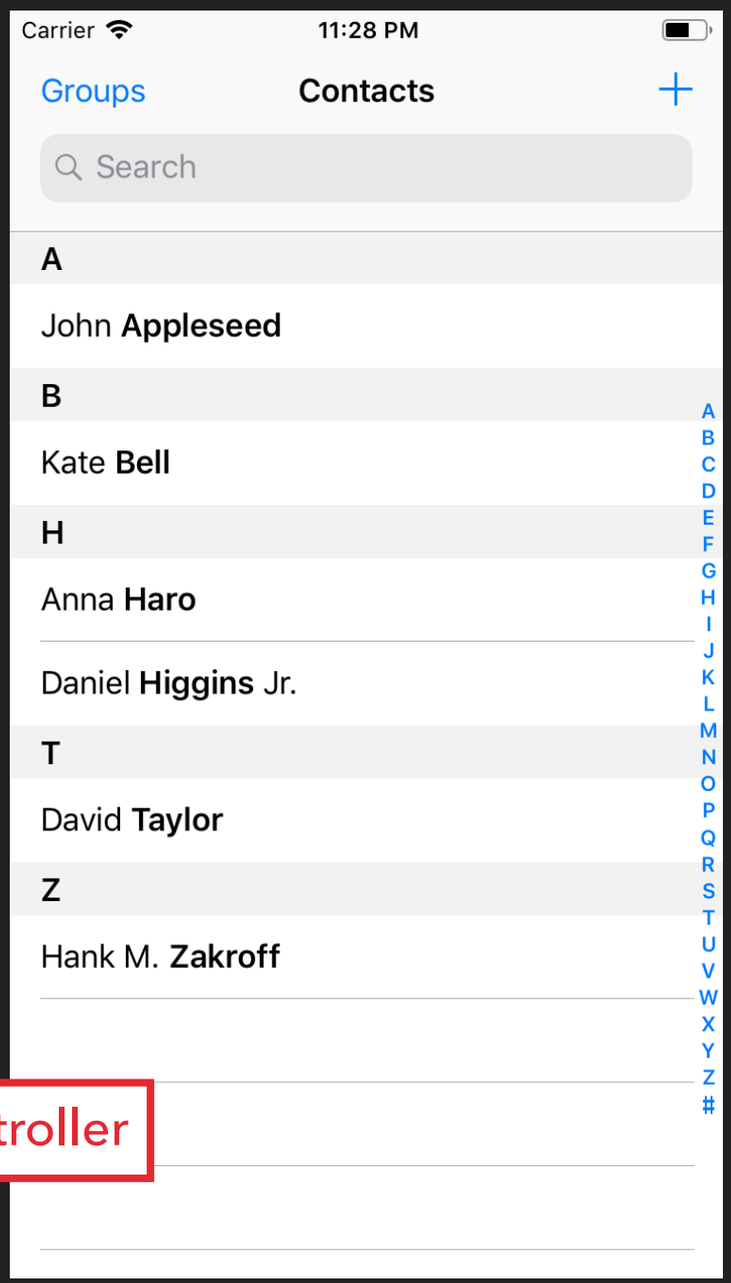
Device	Mode	Portrait orientation	Landscape orientation
12.9" iPad Pro	2/3 split view	Compact width, regular height	Regular width, regular height
	1/2 split view	N/A	Regular width, regular height
	1/3 split view	Compact width, regular height	Compact width, regular height
11" iPad Pro	2/3 split view	Compact width, regular height	Regular width, regular height
	1/2 split view	N/A	Compact width, regular height
	1/3 split view	Compact width, regular height	Compact width, regular height
10.5" iPad Pro	2/3 split view	Compact width, regular height	Regular width, regular height
	1/2 split view	N/A	Compact width, regular height
	1/3 split view	Compact width, regular height	Compact width, regular height
9.7" iPad	2/3 split view	Compact width, regular height	Regular width, regular height

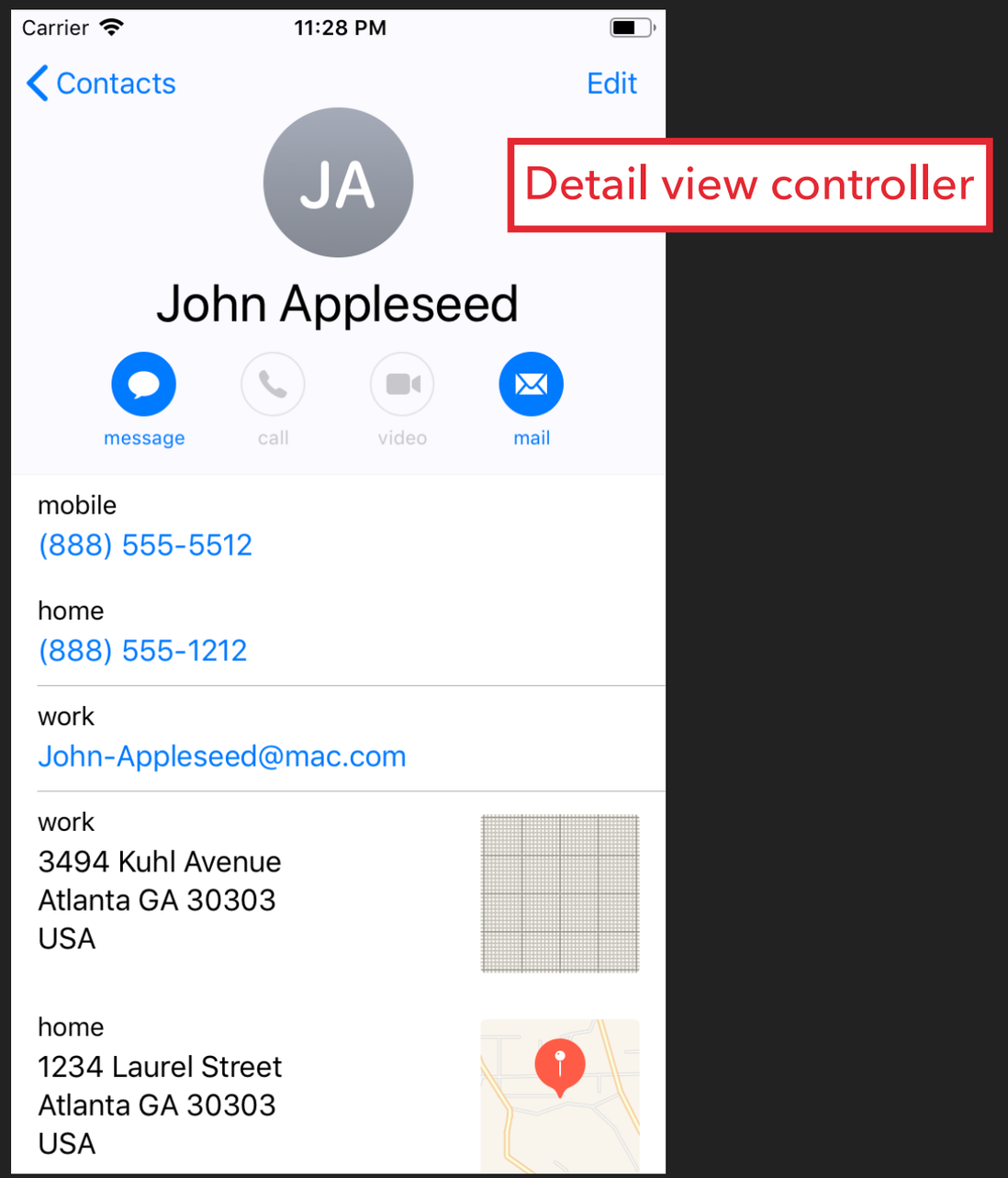
LET'S SEE AN EXAMPLE



Contacts App on iPad Pro Simulator

LET'S SEE AN EXAMPLE



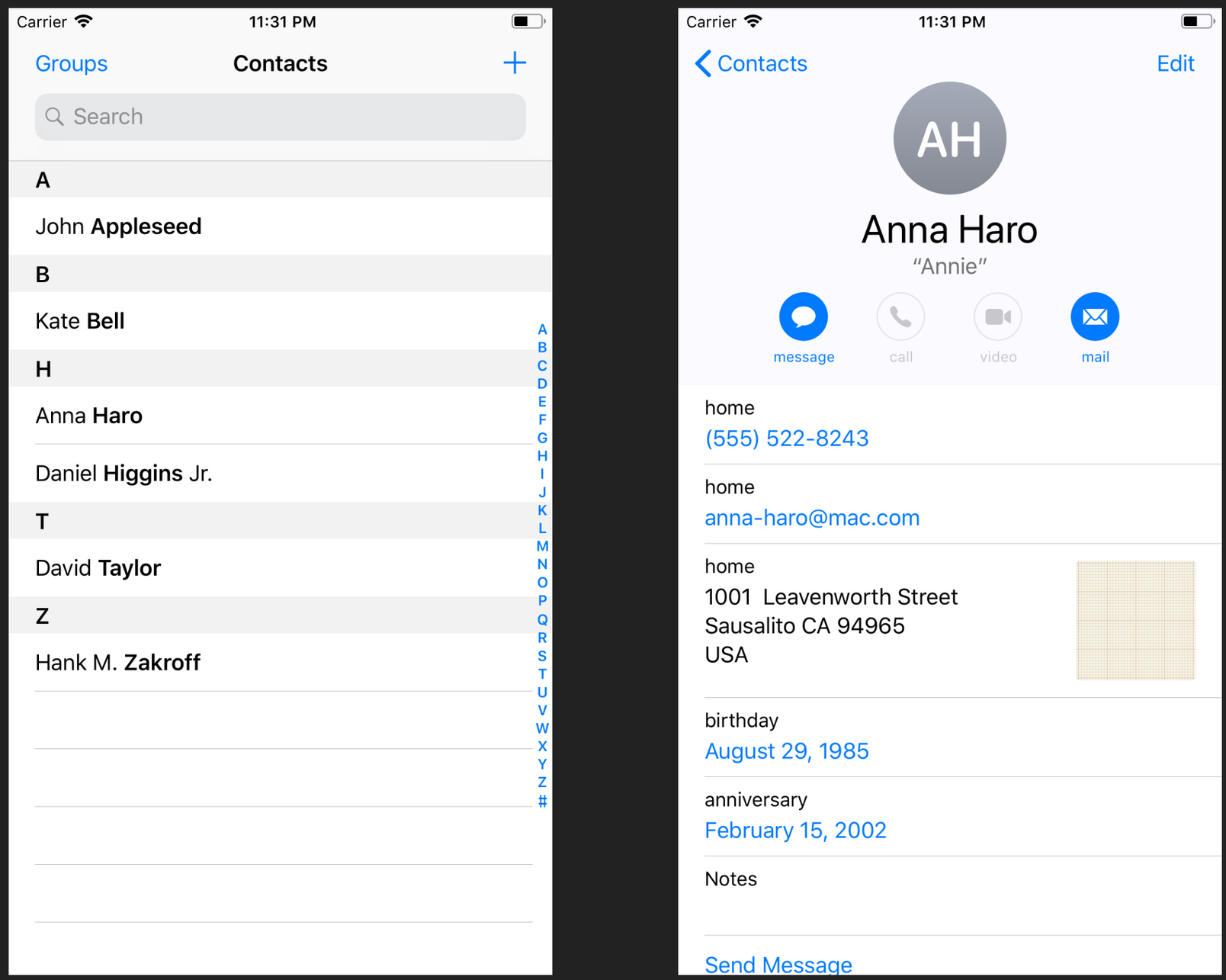


Contacts App on iPhone 8 Simulator

UISPLITVIEWCONTROLLER

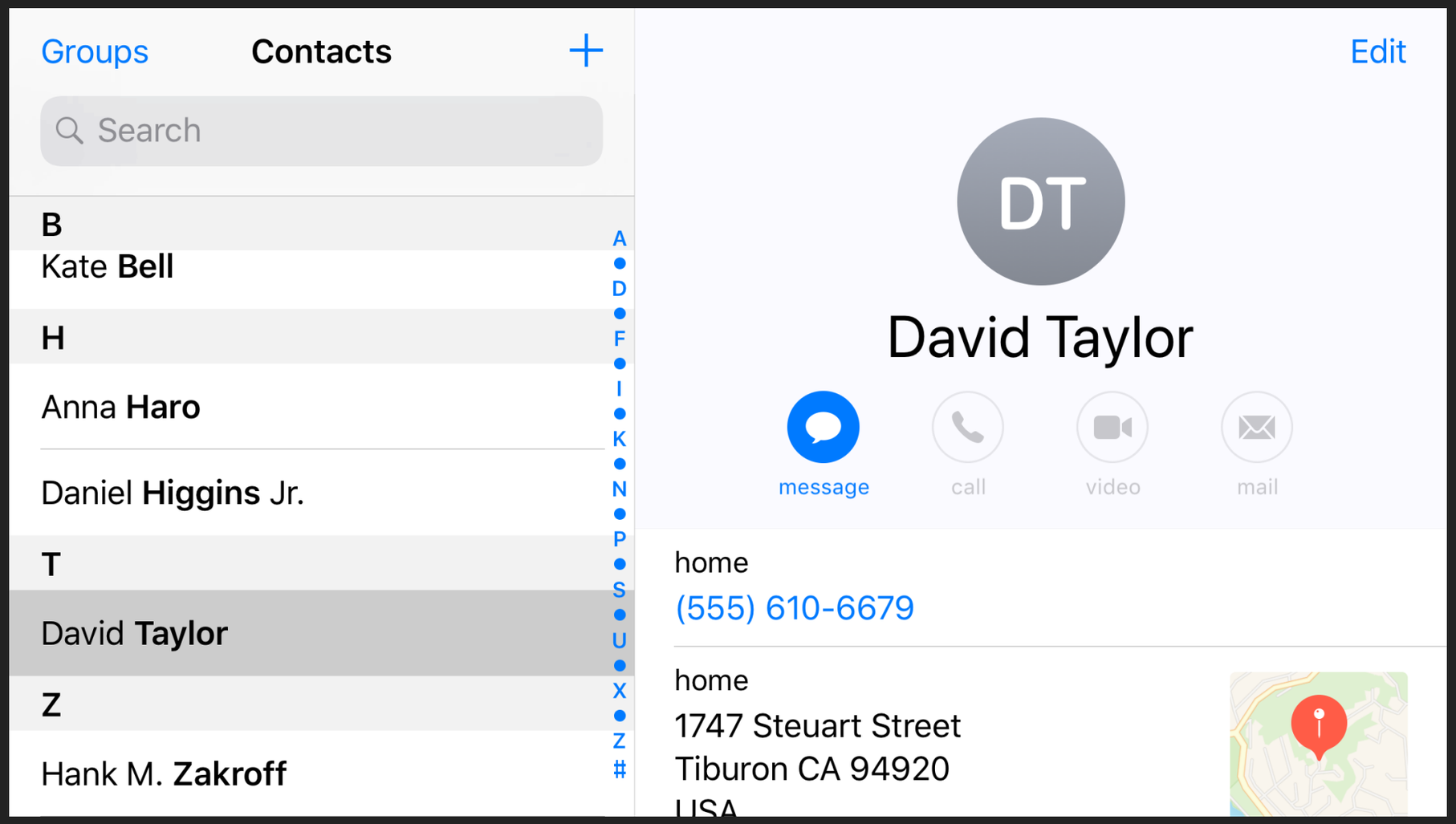
- Compact width –> show master and detail view controllers one at a time
- Regular width –> show master and detail view controllers simultaneously

LET'S SEE AN EXAMPLE



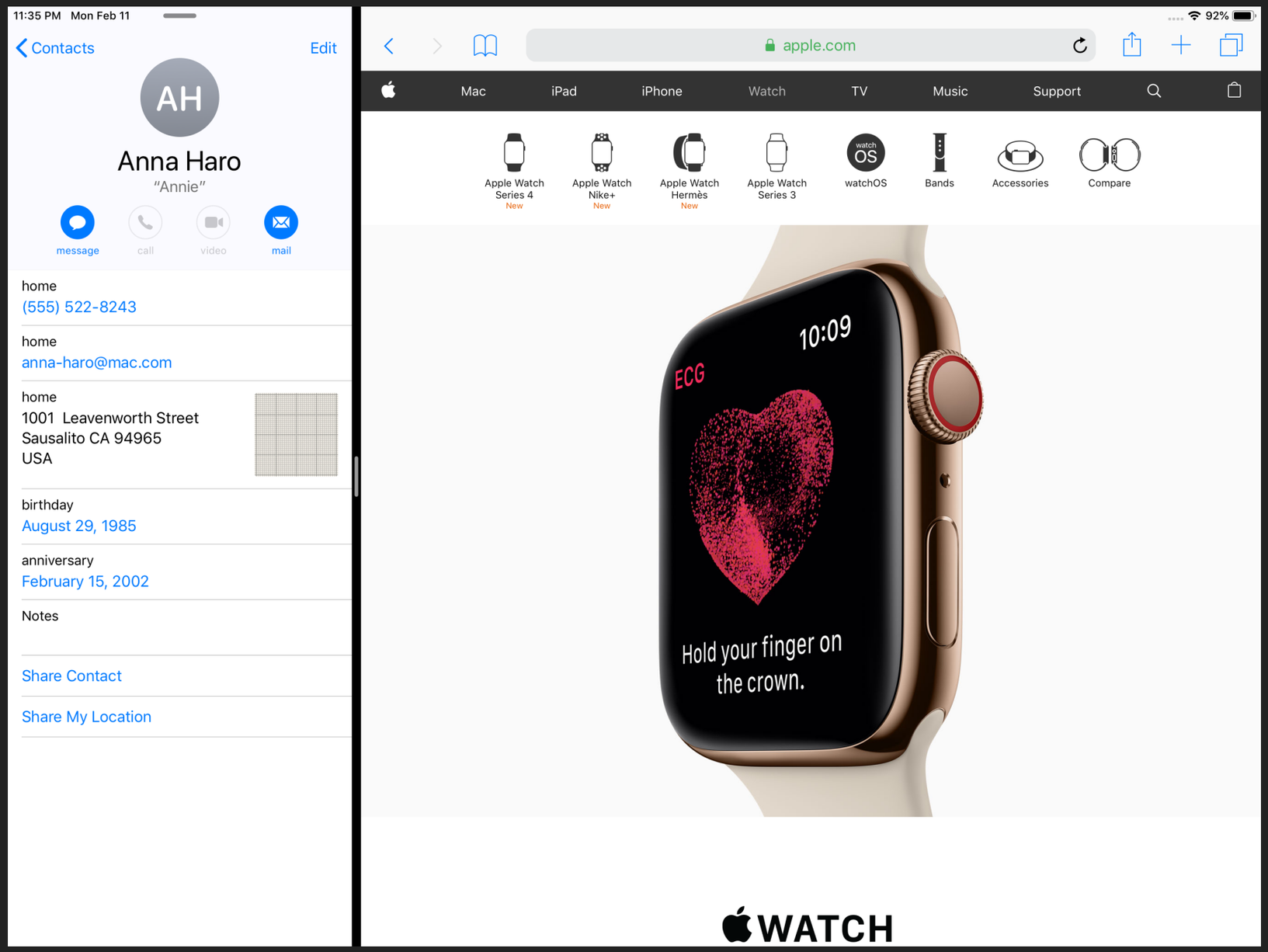
Contacts App on iPhone 8 Plus Simulator in Portrait

LET'S SEE AN EXAMPLE



Contacts App on iPhone 8 Plus Simulator in Landscape

LET'S SEE AN EXAMPLE



Contacts App on iPad Pro Simulator in 1/3 Split View

WHERE IN THE WORLD

LOCATION SERVICES

CORE LOCATION

- The Core Location framework allows developers to access a device's current location
- This relies on three sensors:
  - WiFi
  - Cell
  - GPS

CORE LOCATION

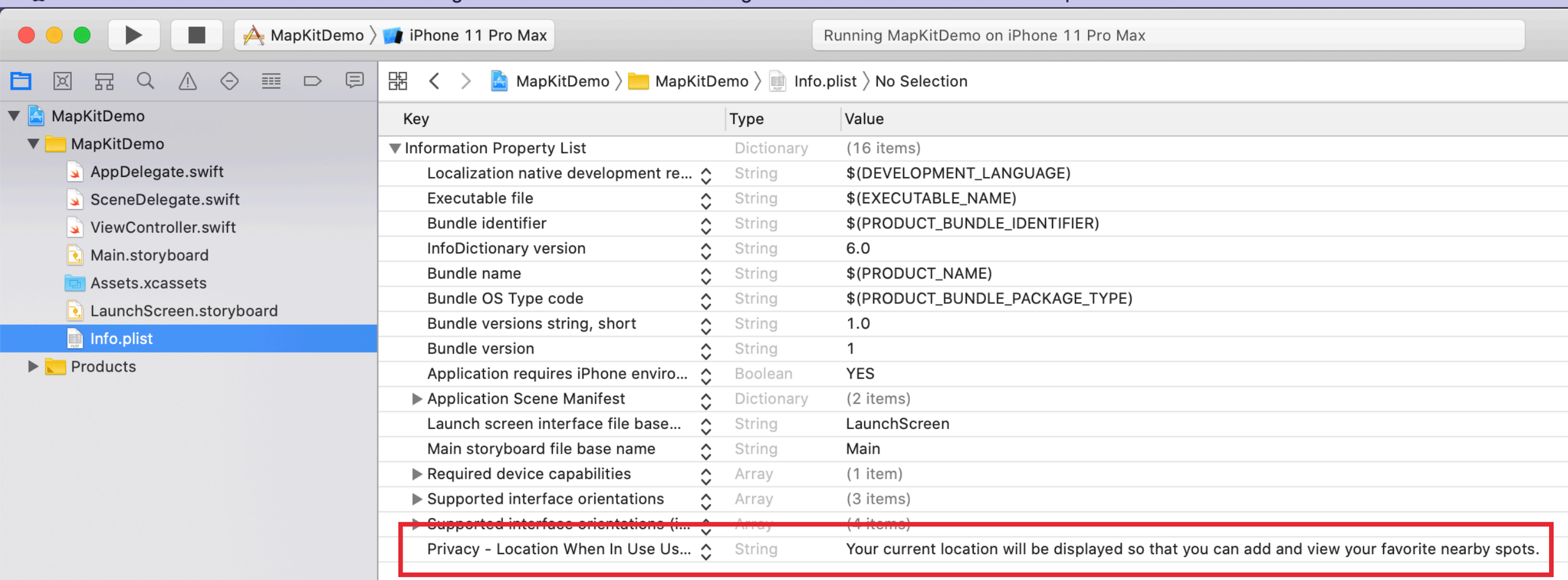
- Create an instance of `CLLocationManager` to start, stop and configure location services
    - When there are updates to a user's location, this object notifies your app through delegate methods
- The delegate must conform to `CLLocationManagerDelegate`

REQUESTING AUTHORIZATION

- Your app must request authorization from the user in order to track the device's location
- There are two types of authorization:
  - Always
  - When in use

REQUESTING AUTHORIZATION

- To request authorization, you must add a location purpose string to your app's Info.plist



REQUESTING AUTHORIZATION

- To request authorization, you must add a location purpose string to your app's Info.plist

Key	Required when:
`NSLocationWhenInUseUsageDescription`	Your app requests When In Use authorization or Always authorization.
`NSLocationAlwaysAndWhenInUseUsageDescription`	Your app requests Always authorization.

REQUESTING AUTHORIZATION

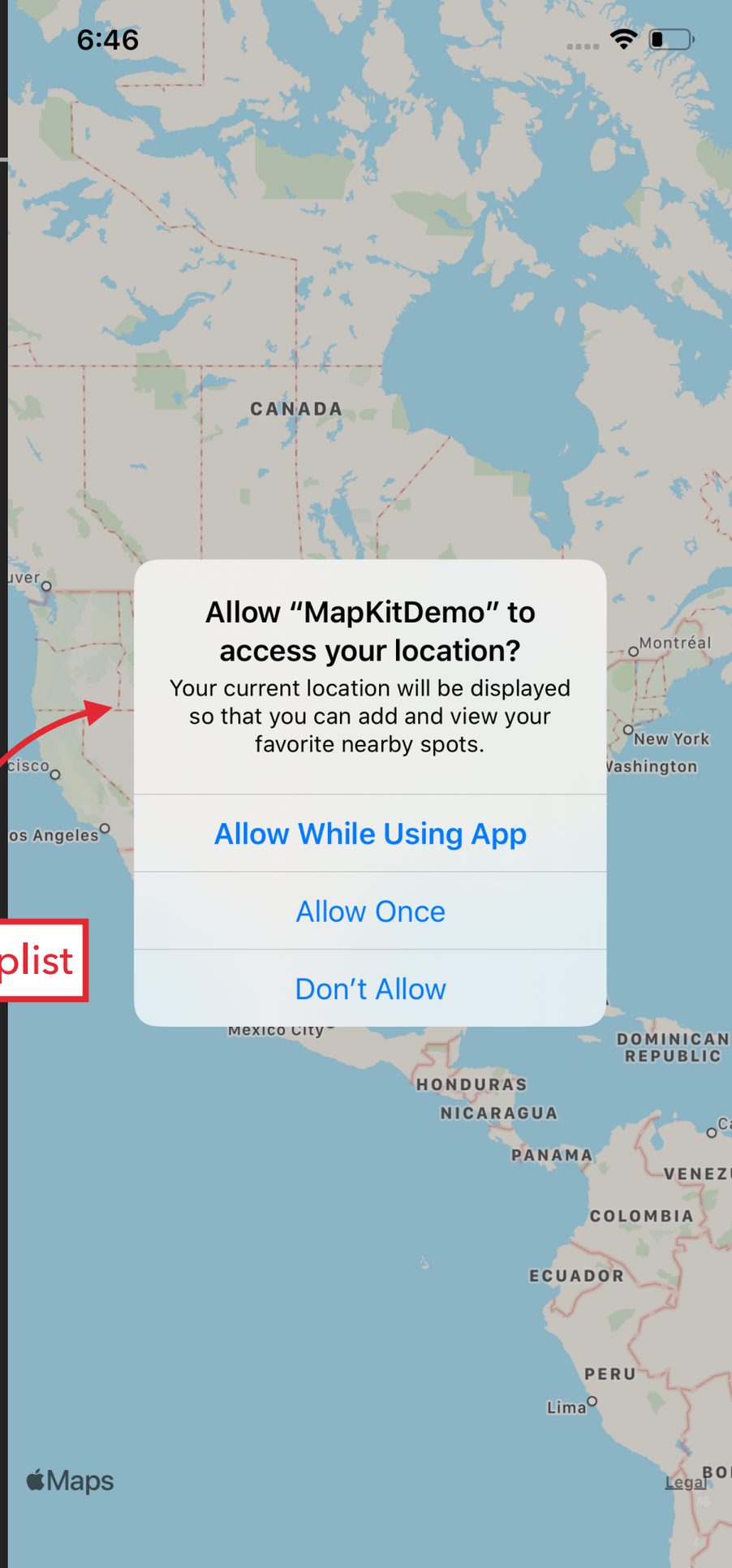
- Then, decide when in your app to request authorization

```
// Request when in use authorization
locationManager.requestWhenInUseAuthorization()

// Request always authorization
locationManager.requestAlwaysAuthorization()
```

REQUESTING AUTHORIZATION

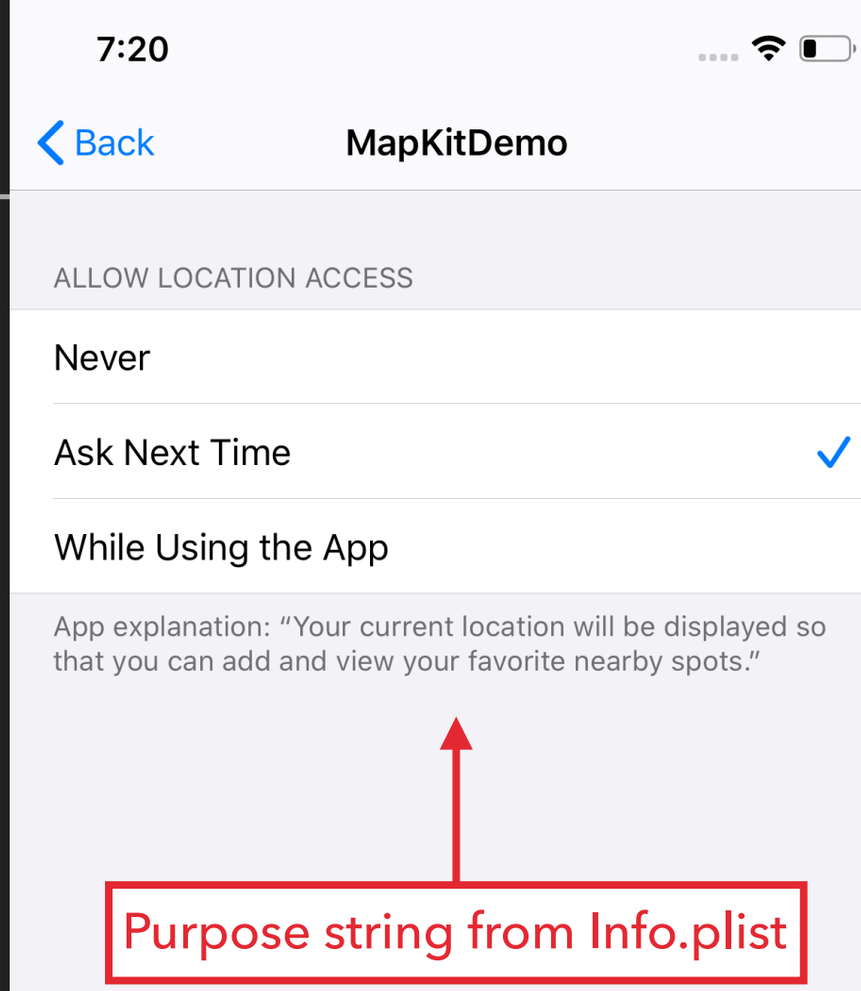
- The user will be presented with this alert
- As of iOS 13, the alert includes the option “Allow Once”

Purpose string from Info.plist



REQUESTING AUTHORIZATION

- The user can change your app's location access at any time in Settings



REQUESTING AUTHORIZATION

- The `didChangeAuthorization` delegate method will be called when authorization status changes

```
func locationManager(_ manager: CLLocationManager,
                     didChangeAuthorization status: CLAuthorizationStatus) {
    switch status {
    case .authorizedAlways, .authorizedWhenInUse:
        print("Authorized!")
    case .notDetermined:
        print("We need to request authorization")
    default:
        print("Not authorized :(")
    }
}
```

LOCATION TRACKING

- Once you have authorization, you can start (and stop) tracking the user’s location

```
// Start tracking the user’s location
locationManager.startUpdatingLocation()

// Stop tracking the user’s location
locationManager.stopUpdatingLocation()
```

LOCATION TRACKING

- Use the `didUpdateLocations` delegate method to receive updates about the user's location

```
func locationManager(_ manager: CLLocationManager,
                     didUpdateLocations locations: [CLLocation]) {

    for location in locations {
        print(location.coordinate)
    }
}
```

LOCATION TRACKING

- Remember to implement `didFailWithError` to receive errors

```
func locationManager(_ manager: CLLocationManager,
                     didFailWithError error: Error) {
    // handle error
}
```

LOCATION TRACKING

- Properties of `CLLocation` include:
  - `coordinate` (latitude and longitude)
  - `altitude` (in meters)
  - `speed` (in meters per second)
  - `course` (degrees clockwise from north)
  - `timestamp`

LOCATION TRACKING

- Not all apps require the same level of granularity from location tracking
- Consider the difference between apps that provide:
  - Fitness tracking
  - Food delivery
  - Weather updates

LOCATION TRACKING

- Developers can determine the granularity of location tracking by setting properties on `CLLocationManager`
  - desiredAccuracy
  - distanceFilter
  - pausesLocationUpdateAutomatically
  - activityType

LOCATION MONITORING

- Alternatively, use location monitoring instead of tracking
- Types of monitoring:
  - Significant location change
  - Visit monitoring
  - Region monitoring
  - Geofenced local notifications

REQUEST LOCATION

- If your app only needs to fetch the user's current location (without tracking or monitoring), you can issue a one-off request

```
locationManager.requestLocation()
```

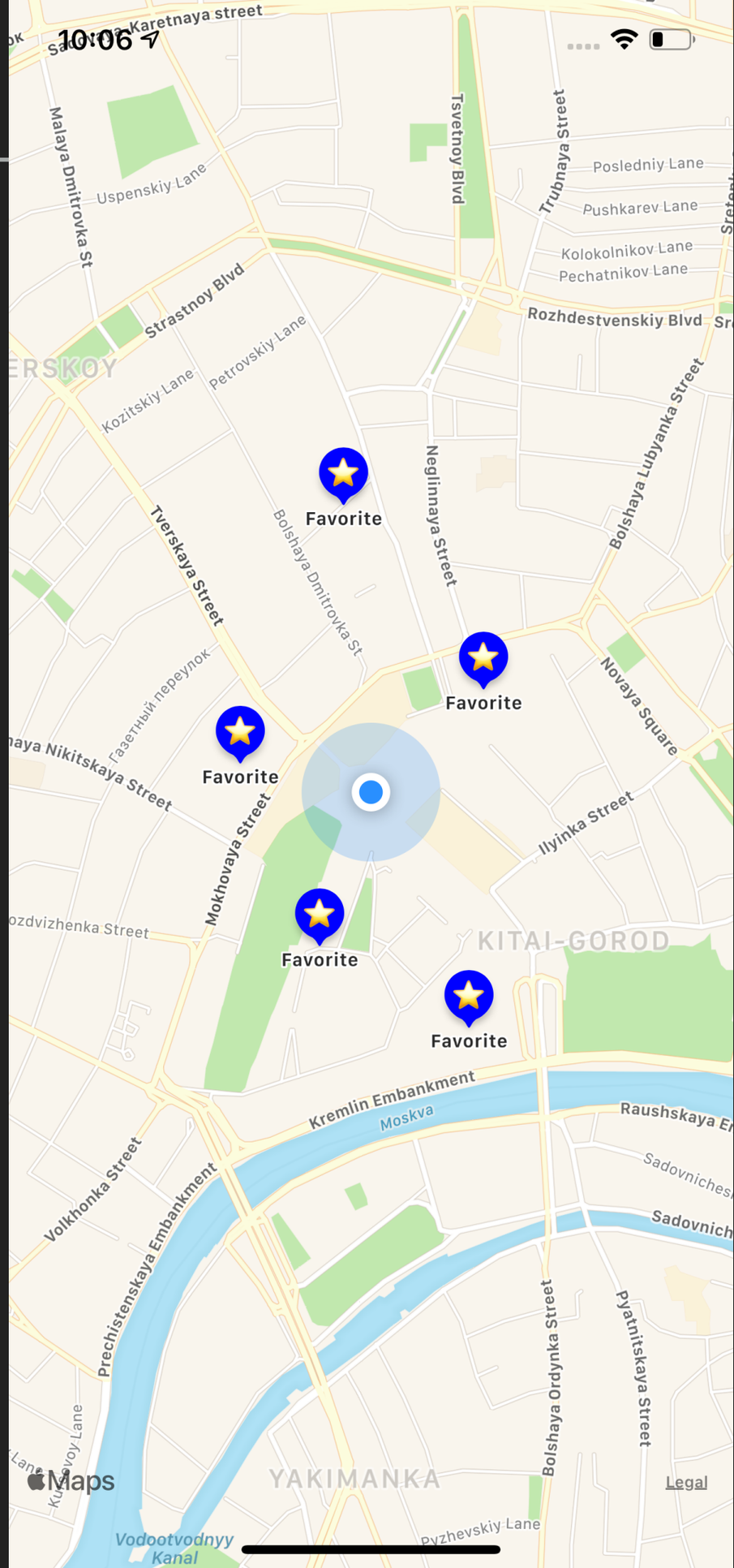
- The `didUpdateLocations` method will be called exactly once

WHERE IN THE WORLD

MAP KIT

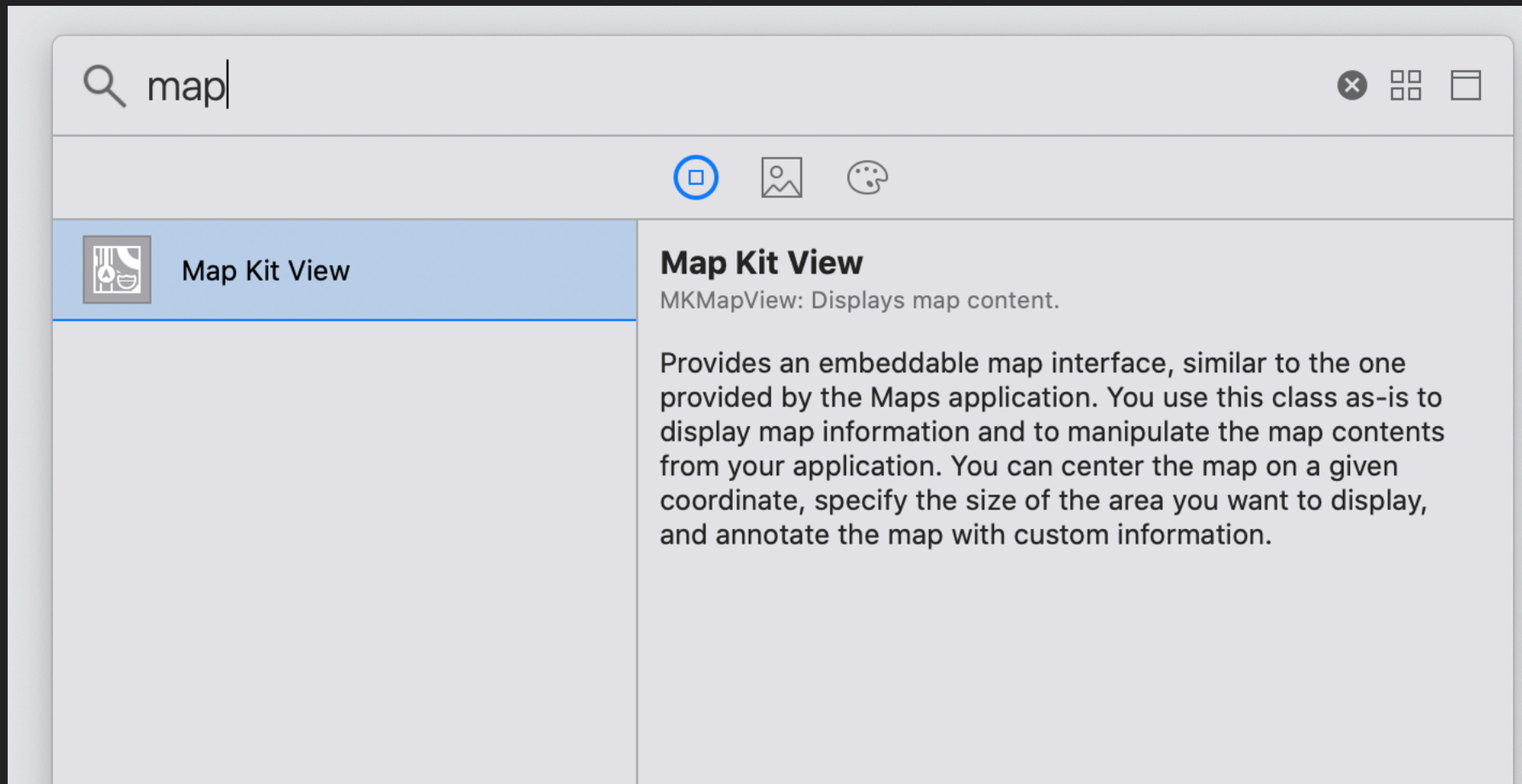
MAP KIT

- The MapKit framework works with Core Location to allow you to display maps in your application



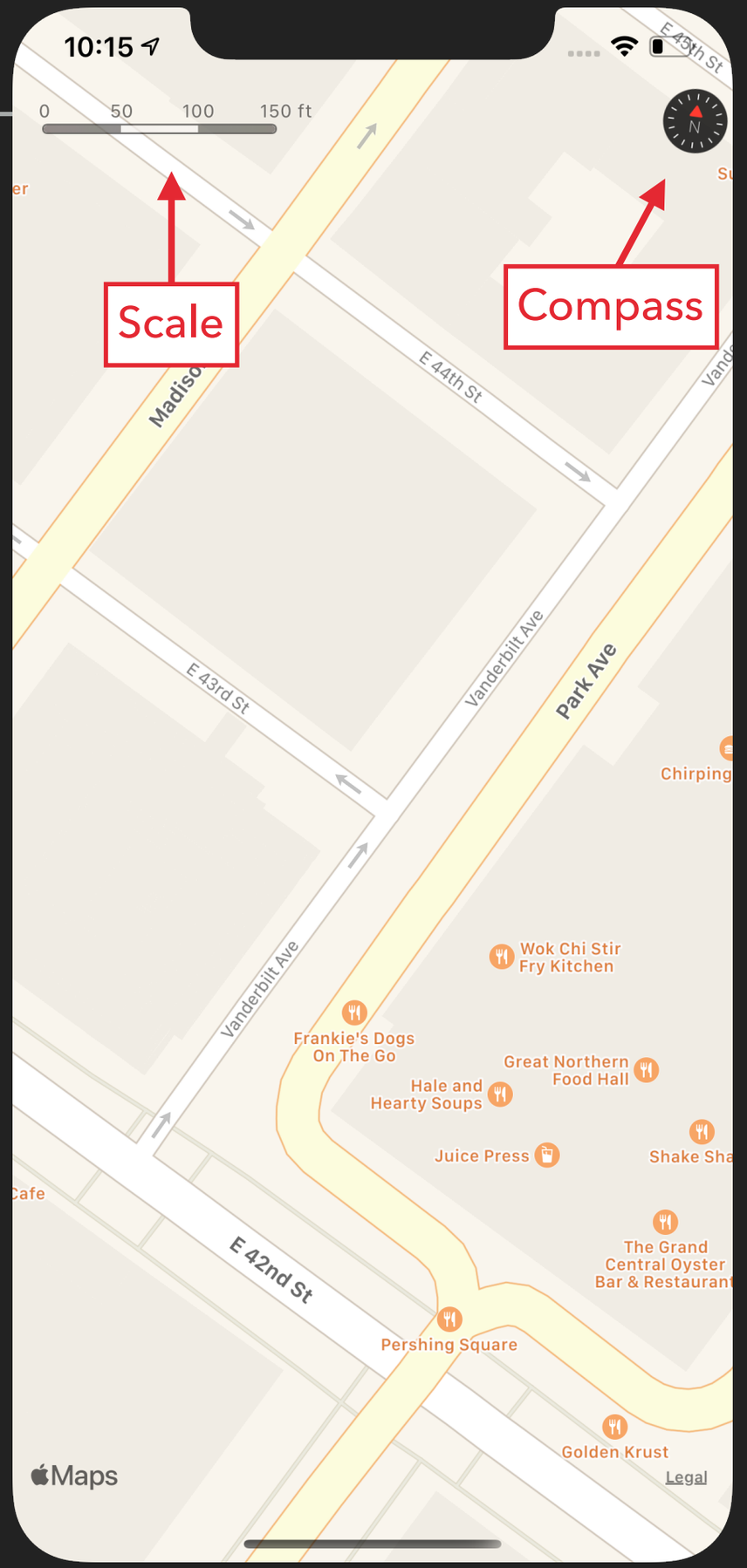
MAP KIT

- An instance of `MKMapView` displays a map on screen
  - Can be added programmatically or in storyboard



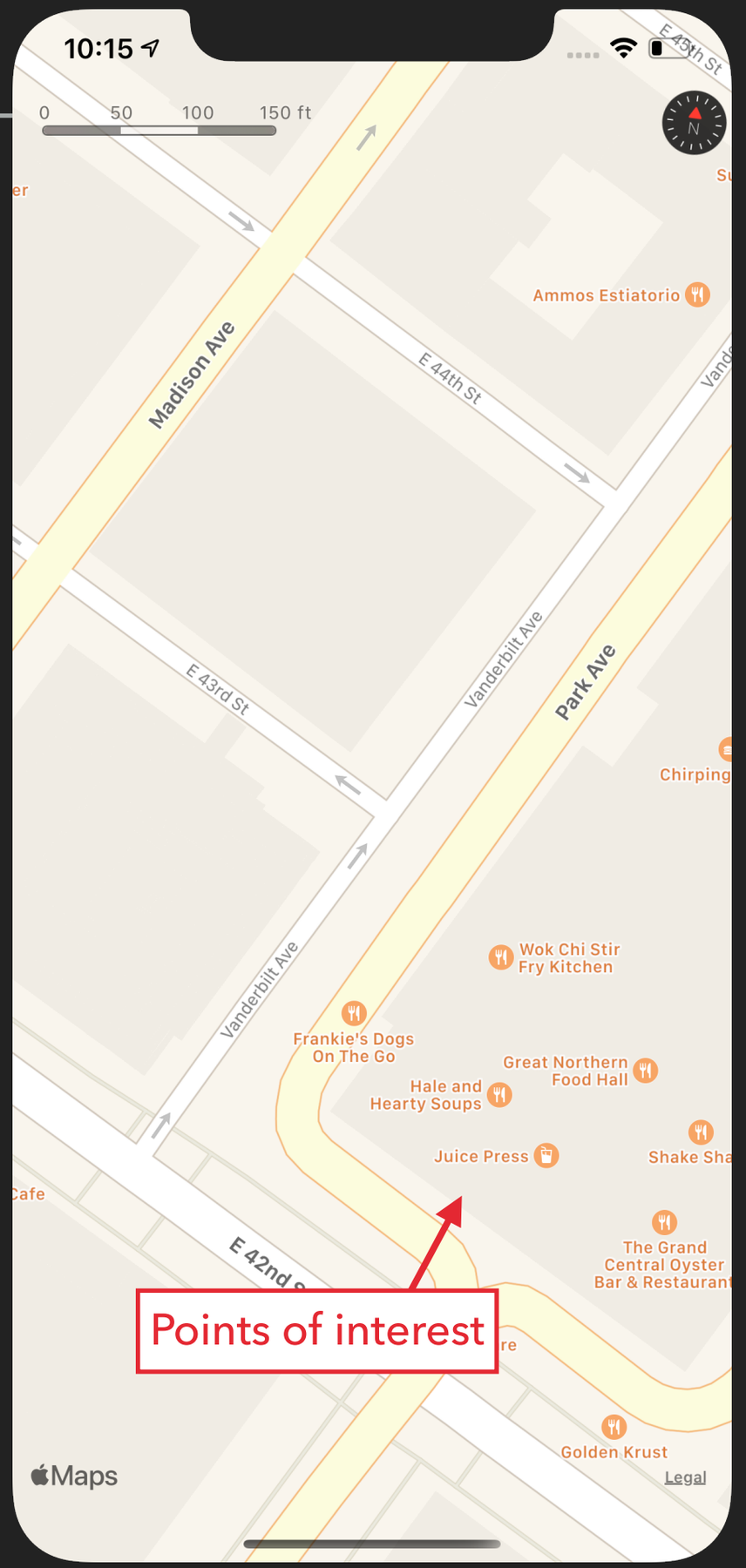
MAP KIT

- The map view can be configured with properties such as:
  - `showsCompass`
  - `showsScale`
  - `showsTraffic`



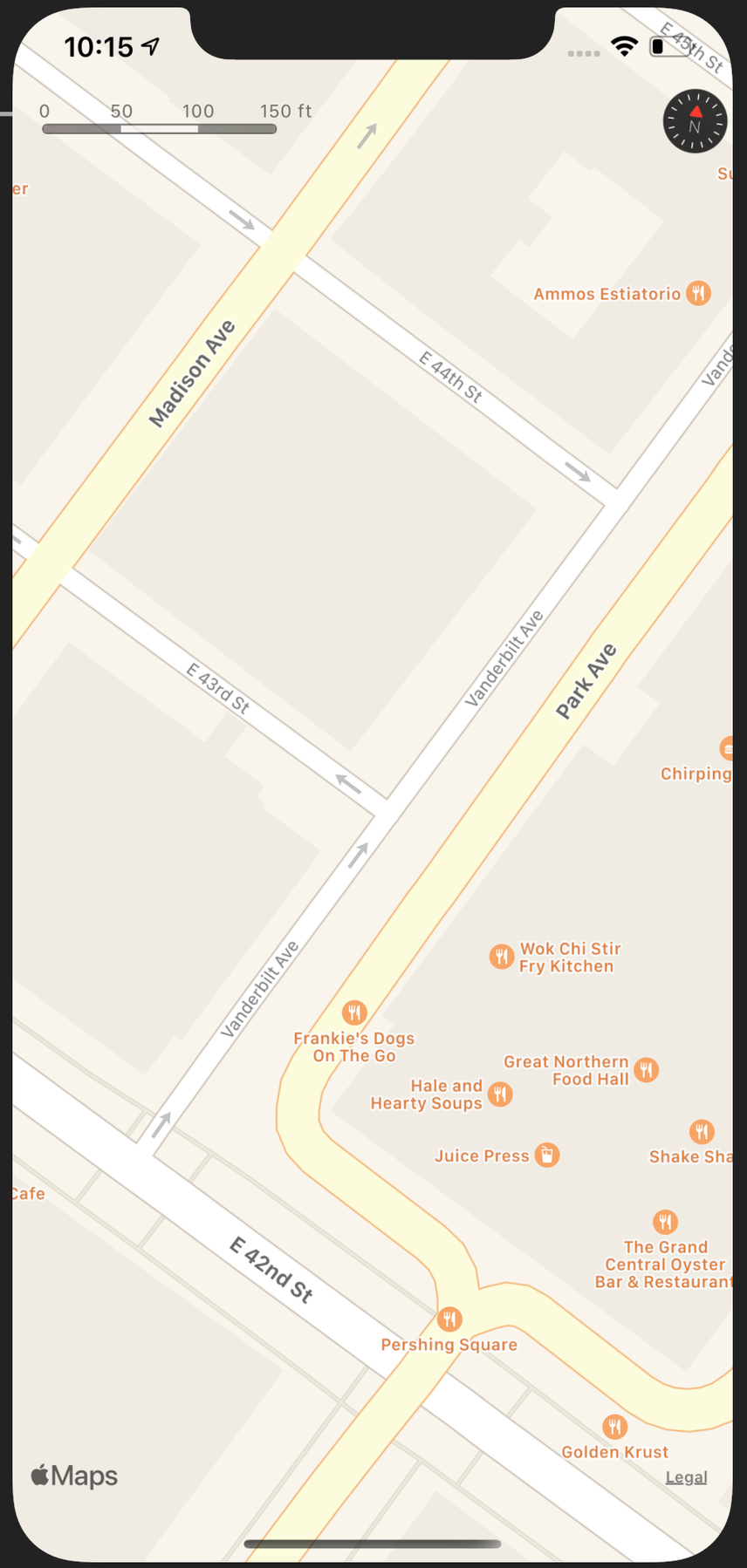
MAP KIT

- The map view can be configured with properties such as:
  - `pointsOfInterestFilter`
  - This is an array containing categories such as airport, gas station, and restaurant



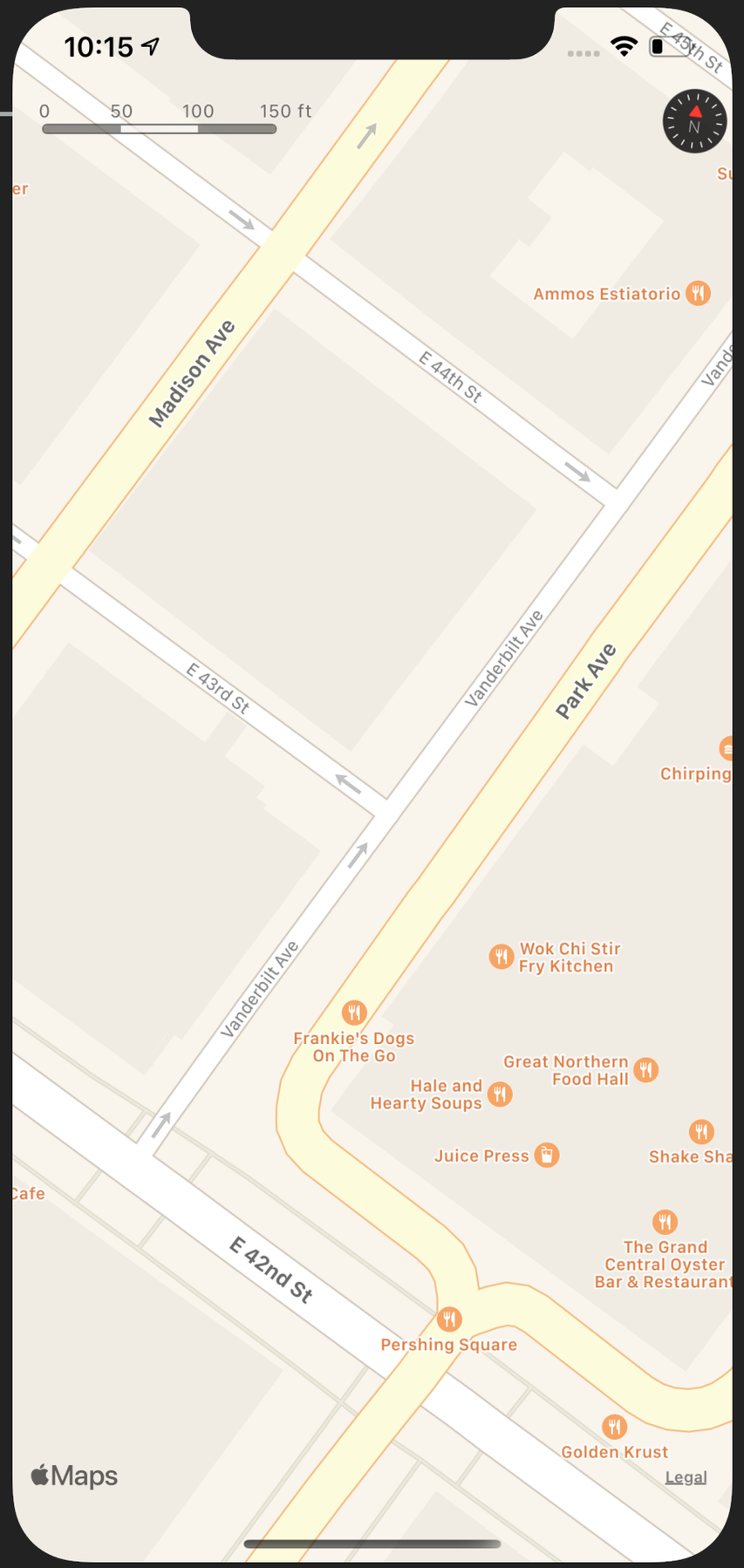
MAP KIT

- The map view can be configured with properties such as:
  - `region`
  - The center and span of the area the map view should display



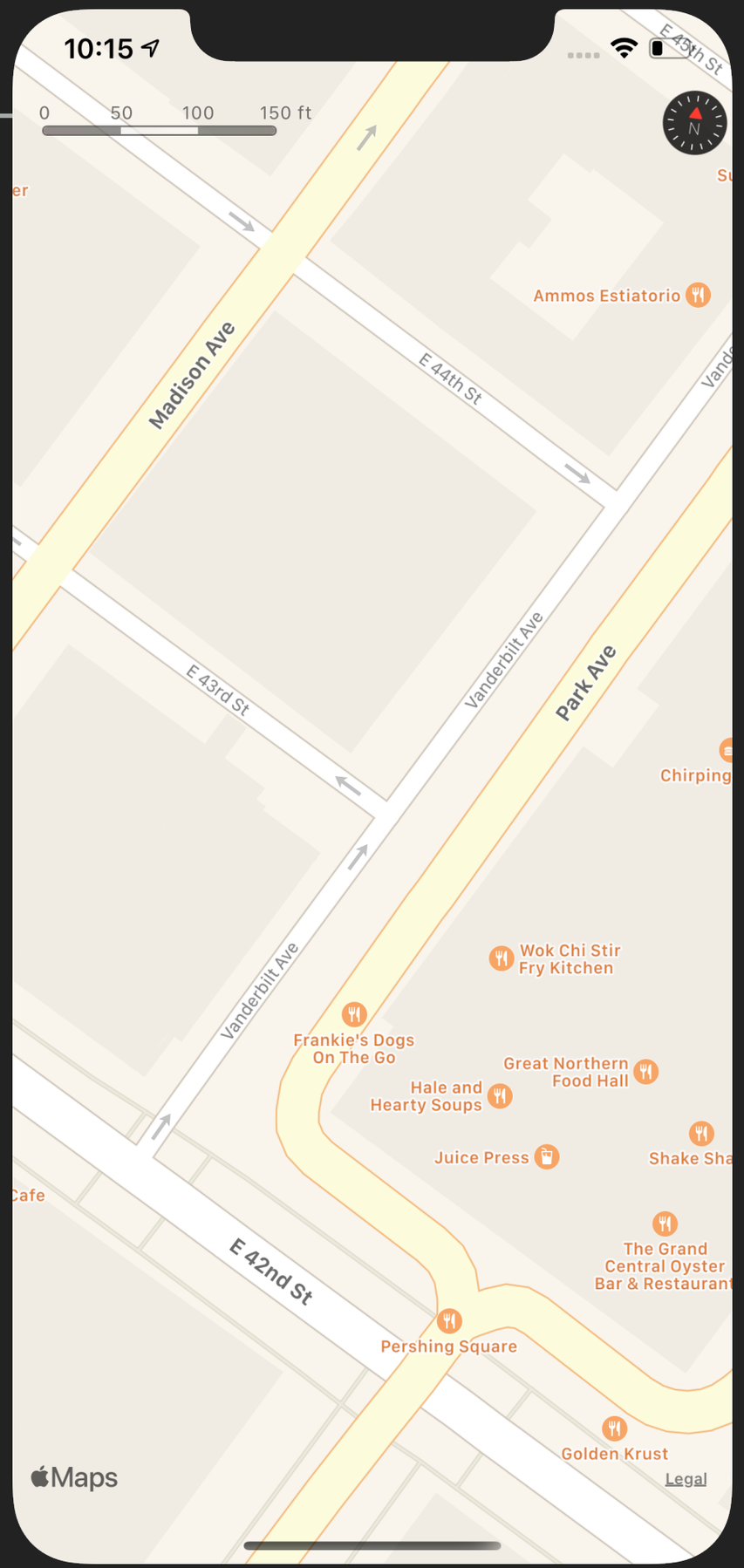
MAP KIT

- The map view can be configured with properties such as:
  - `cameraZoomRange`
  - `cameraBoundary`



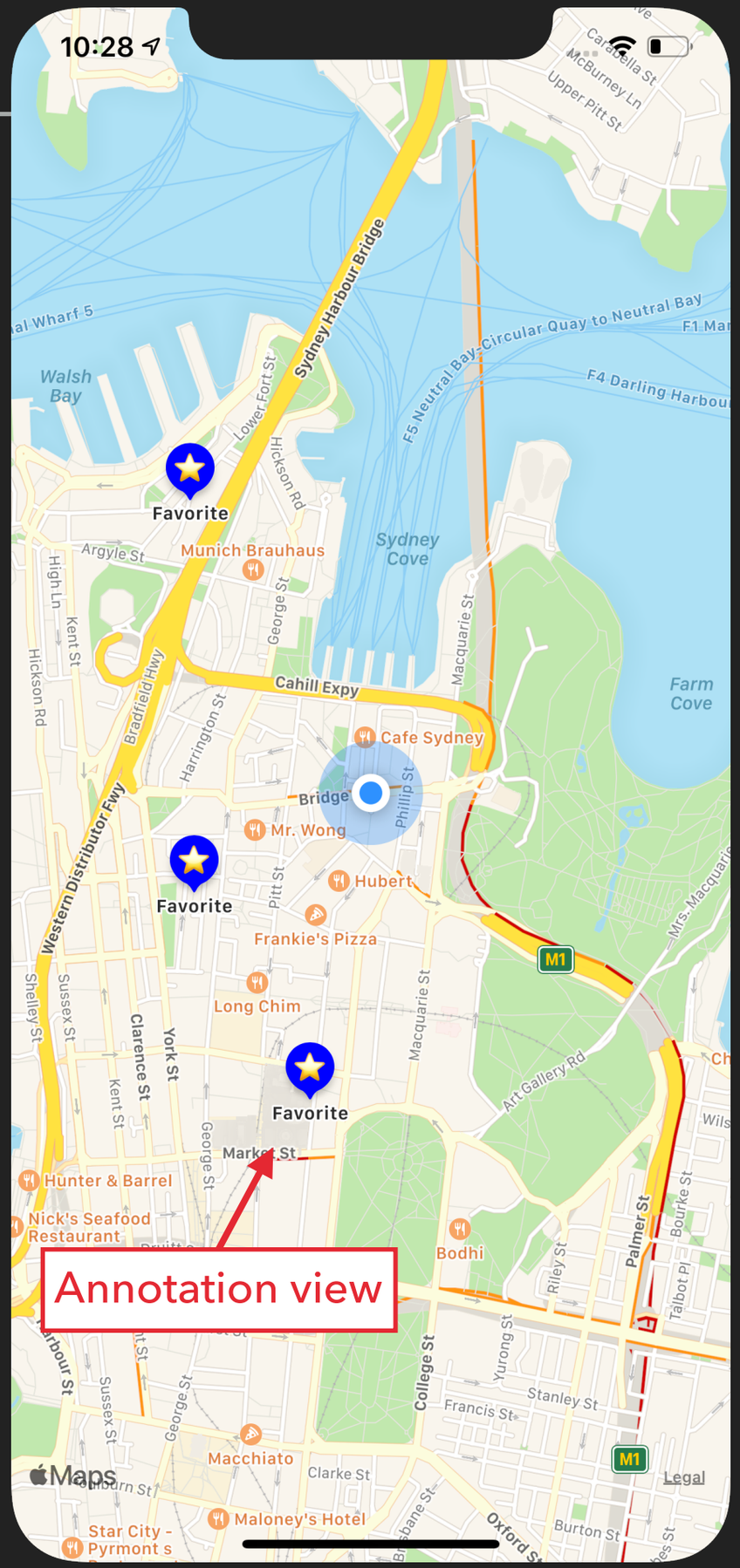
MAP KIT

- The map view's delegate receives updates with as the map loads and as its visible region changes
- The delegate must conform to `MKMapViewDelegate`



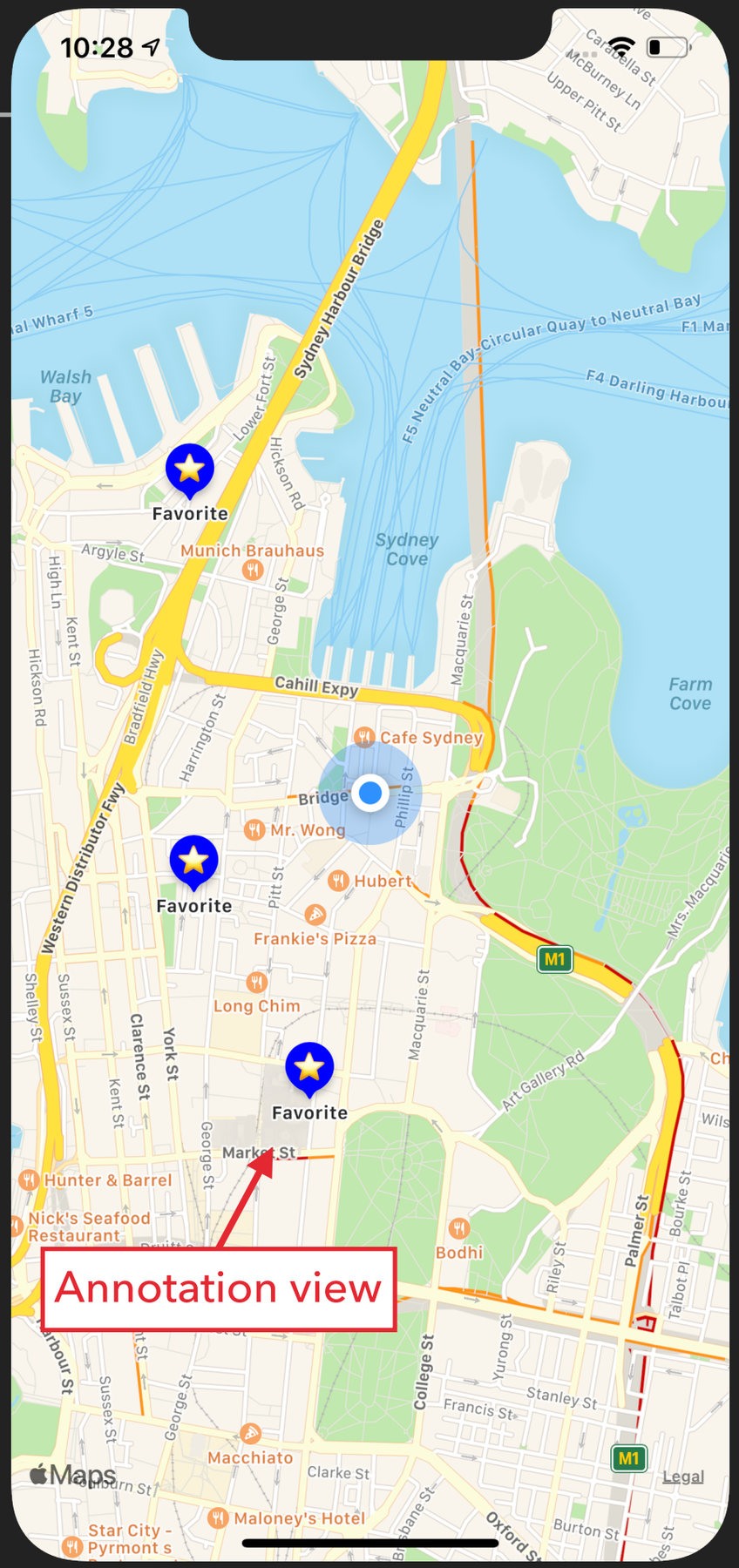
MAP KIT

- Map views can display annotation views that call attention to certain spots



MAP KIT

- A map view may have hundreds or thousands of possible annotation views.
- Loading them all at the same time might put a strain on memory. How do you imagine the system deals with this?



MAP KIT

- The `viewFor` delegate method passes in an annotation and expects an annotation view

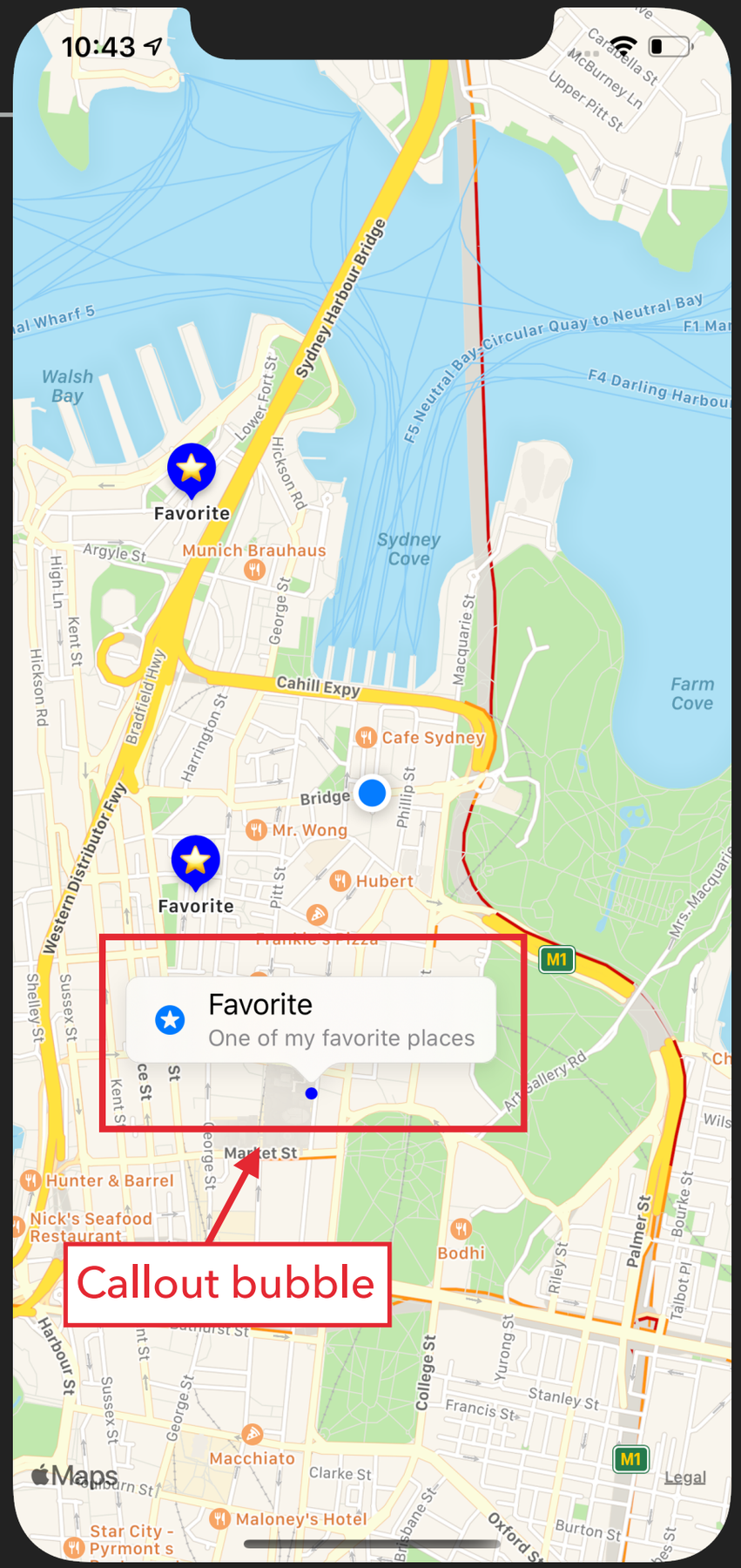
```
func mapView(_ mapView: MKMapView,
             viewFor annotation: MKAnnotation) -> MKAnnotationView? {

    guard let annotationView = mapView.dequeueReusableAnnotationView(
        withIdentifier: id, for: annotation) as? MKMarkerAnnotationView else {
        return nil
    }

    // configure the annotation view

    return annotationView
}
```

ANNOTATION VS ANNOTATION VIEW

- An `MKAnnotation` represents the data
- Its properties include:
  - `coordinate`
  - `title`
  - `subtitle`

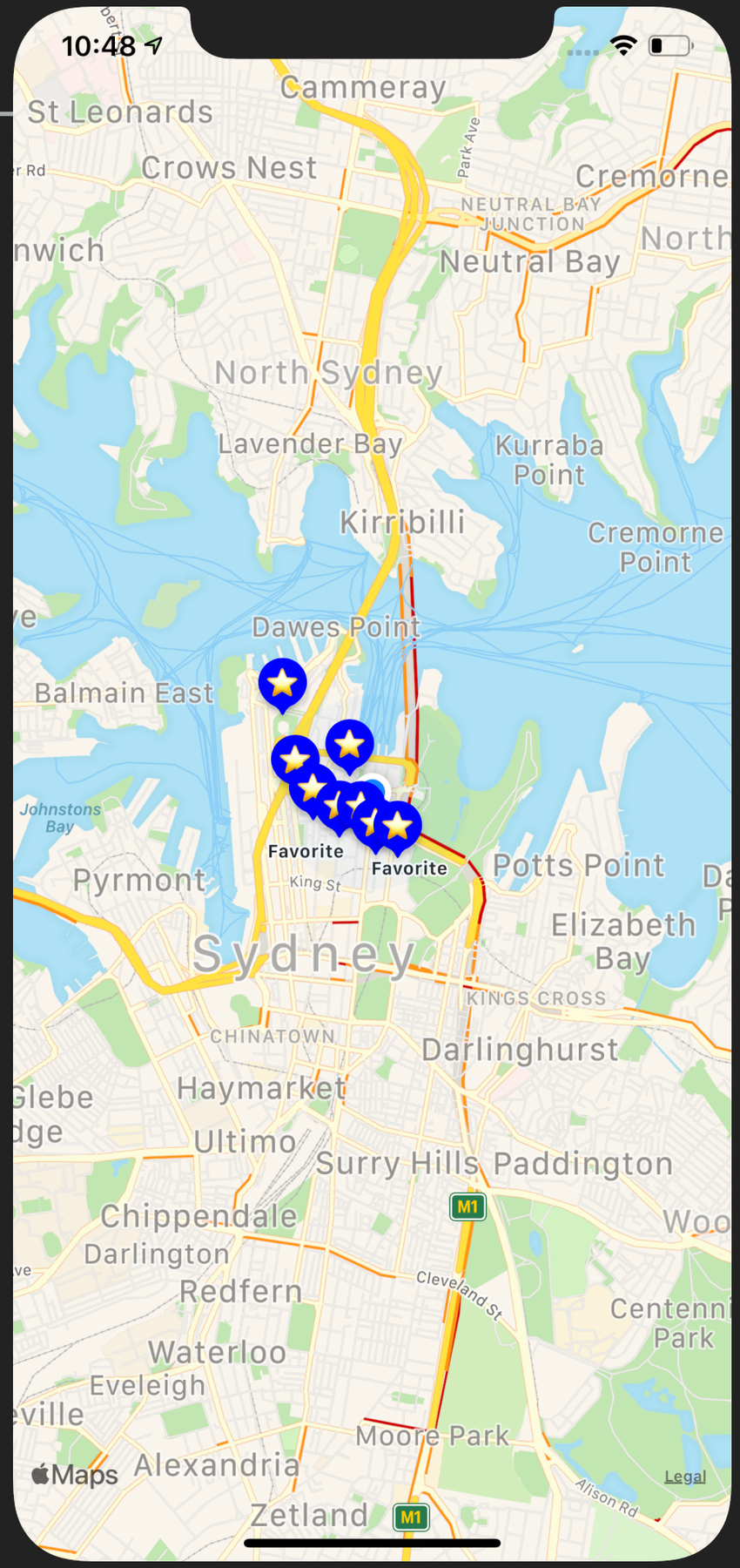
ANNOTATION VS ANNOTATION VIEW

- An `MKAnnotationView` represents the view displayed on the map
- It displays an image and optionally a callout bubble with text and accessory views



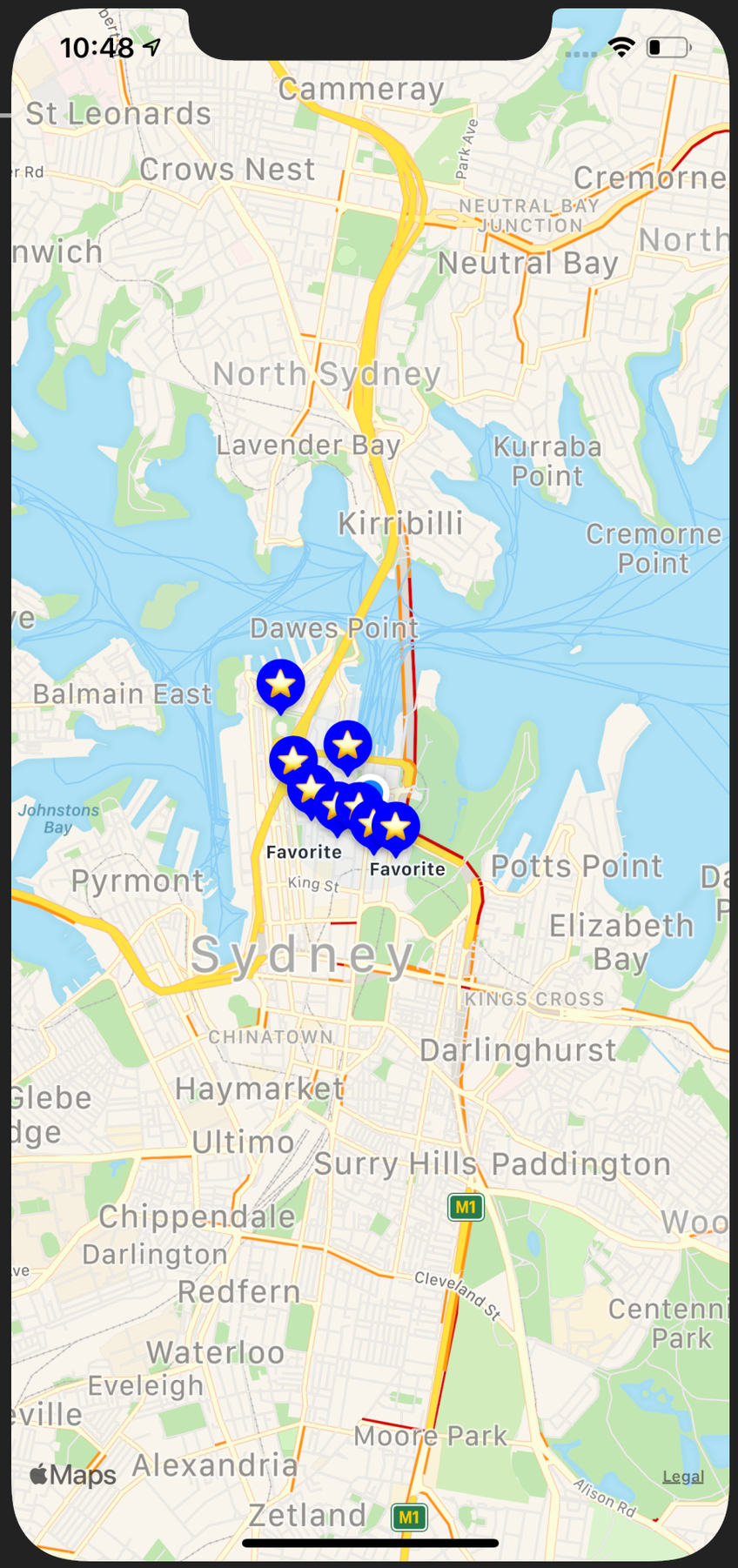
CLUSTERING

- Sometimes there are groups of annotation views that are too close together
- This causes the UI to appear cluttered and could degrade performance



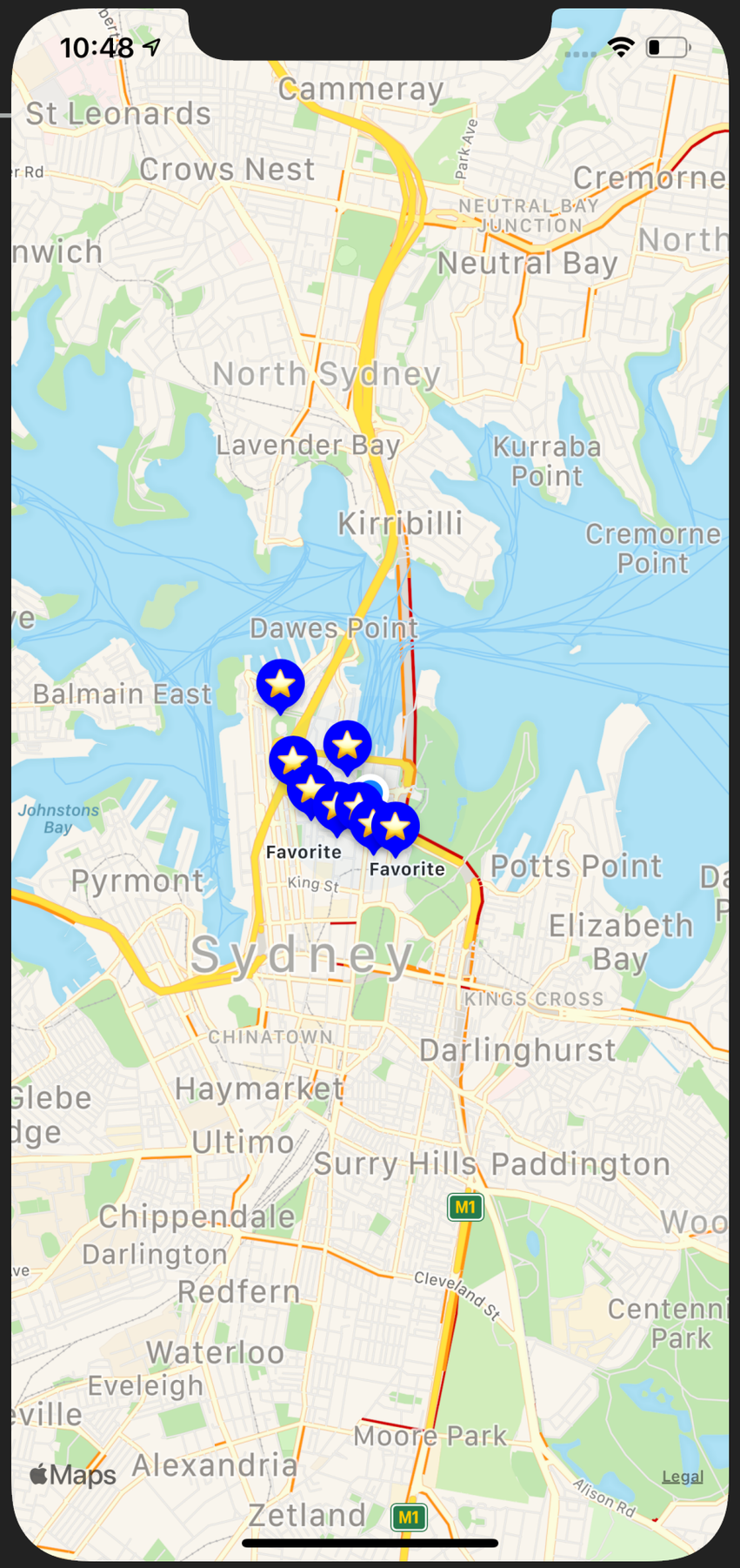
CLUSTERING

- Before iOS 11, developers would manually detect when too many annotation views appeared on top of each other
- They would replace groups of annotation views with a single annotation view



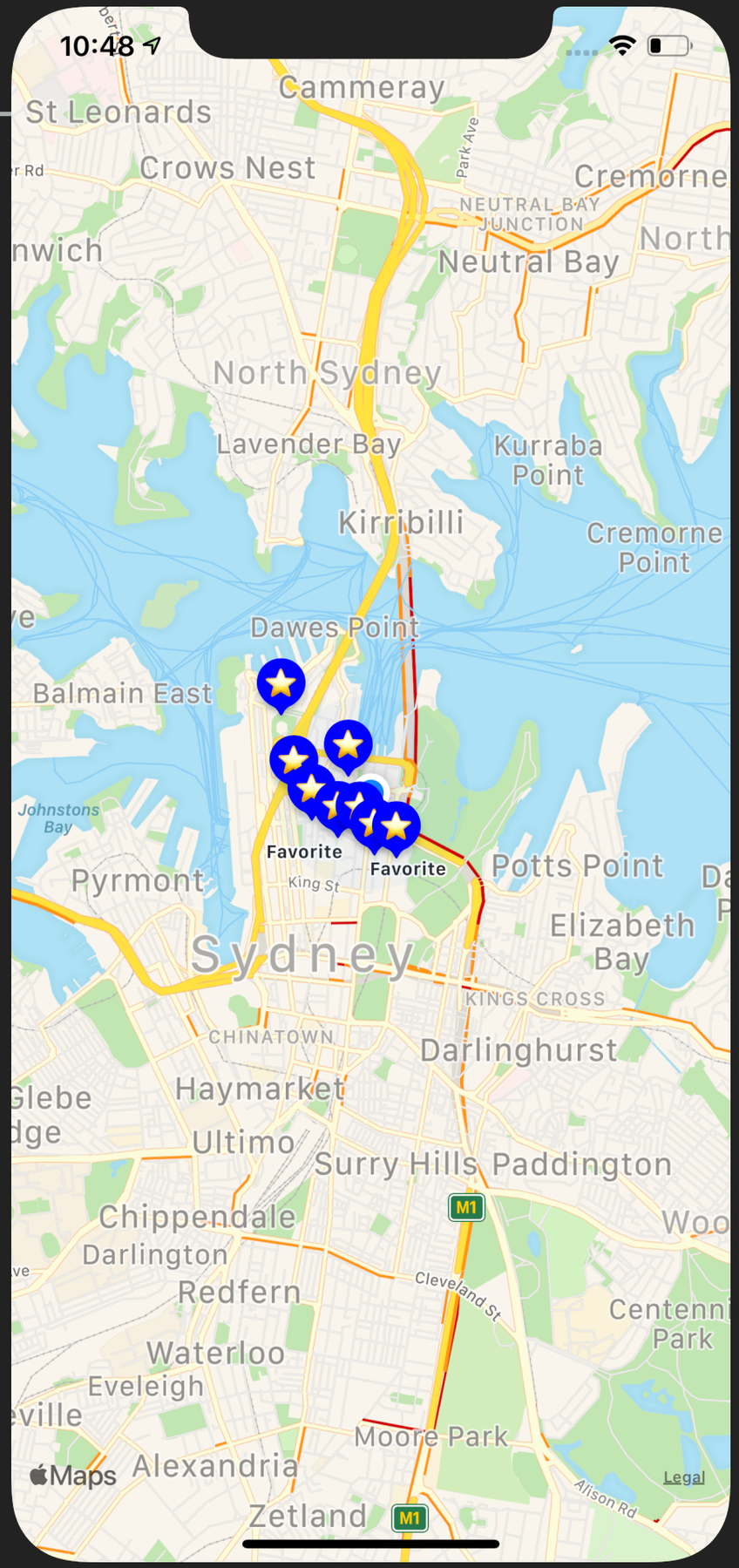
CLUSTERING

- Starting in iOS 11, annotation views can automatically show and hide themselves as the view becomes crowded
- Developers can use special annotation views to indicate a cluster



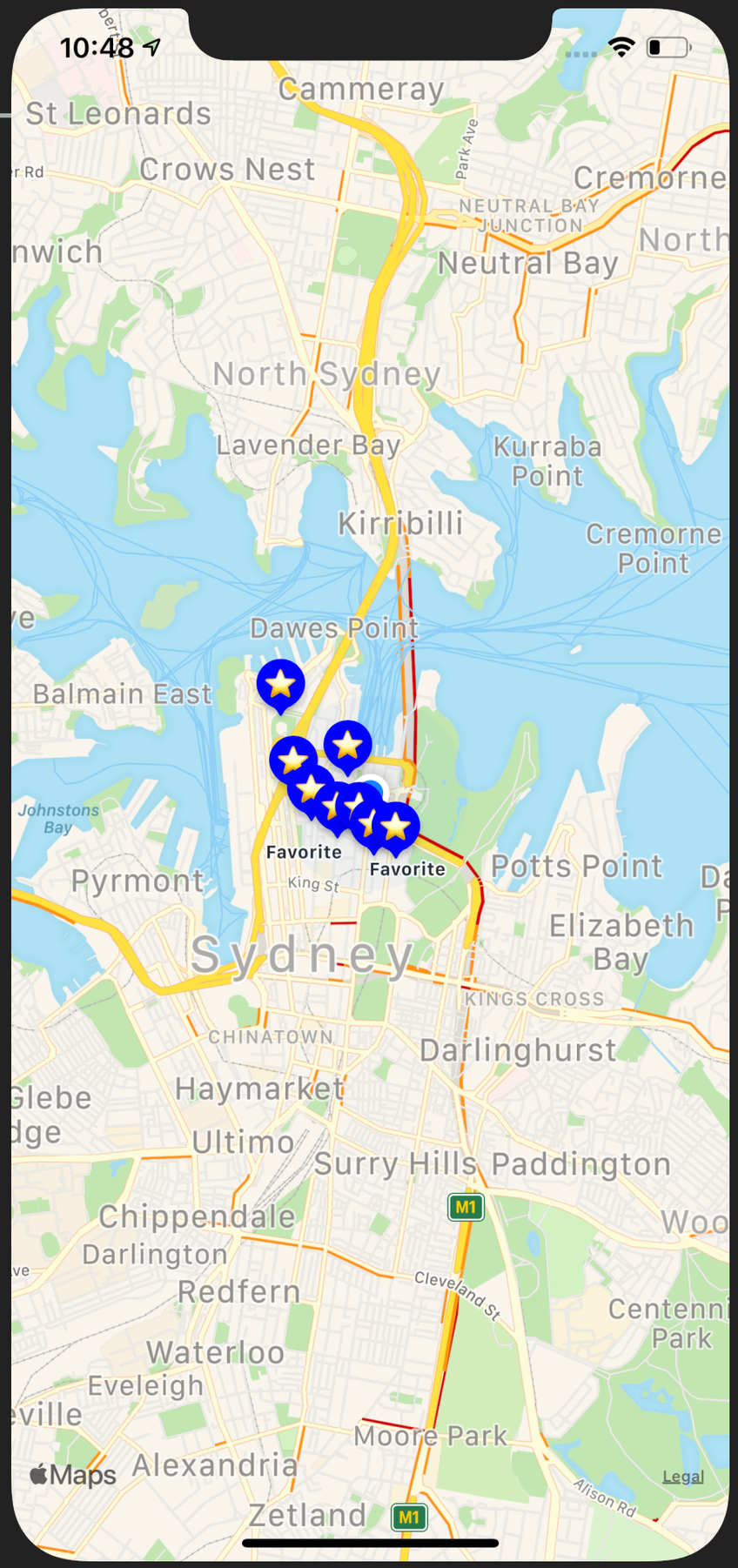
CLUSTERING

- The `priority` of an annotation view determines whether it can be hidden
  - 1000 = required (never hidden)
- Annotation views with lower priorities will be hidden first



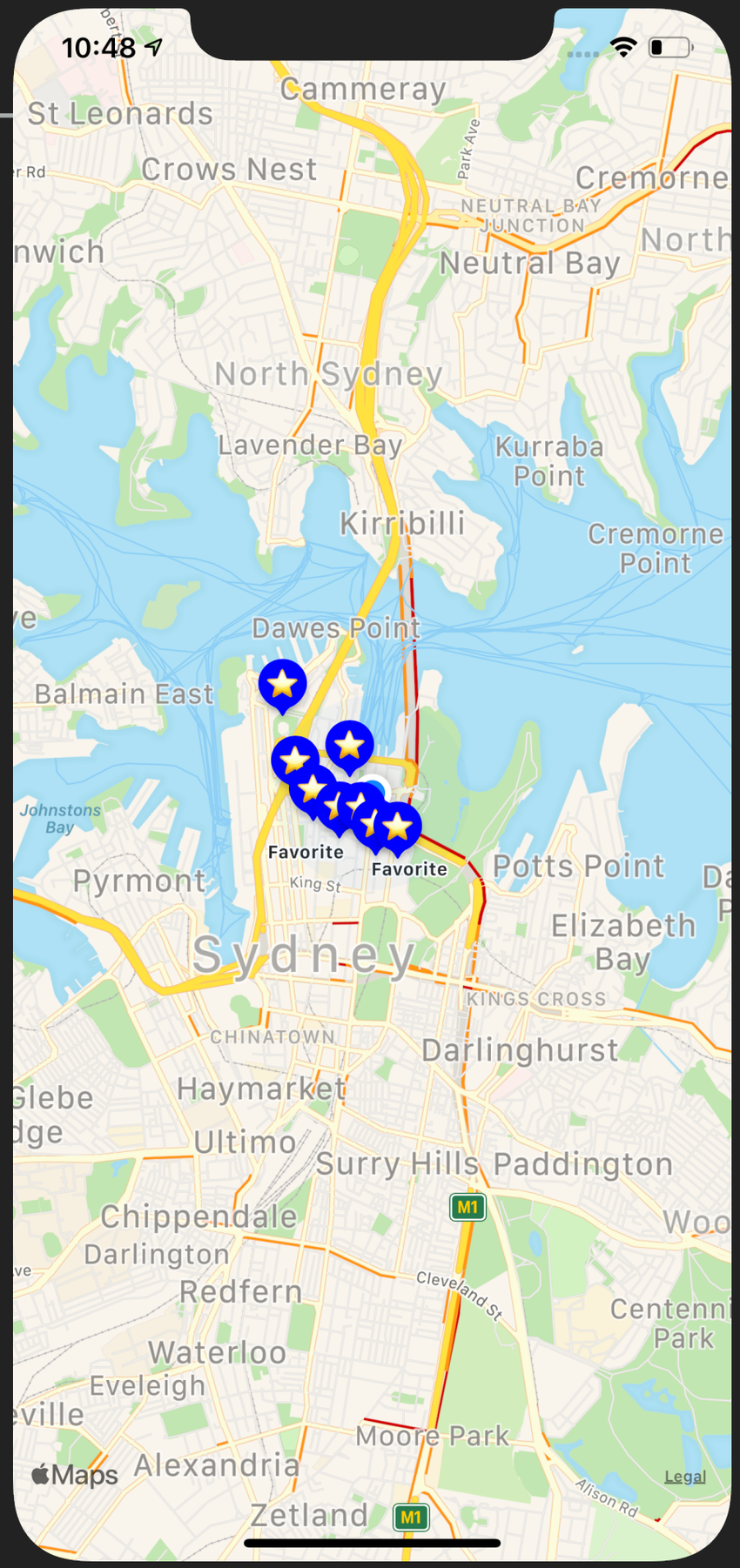
CLUSTERING

- Annotation views with the same `clusteringIdentifier` can be clustered together
- Annotation views without a clustering identifier won't participate in clustering



CLUSTERING

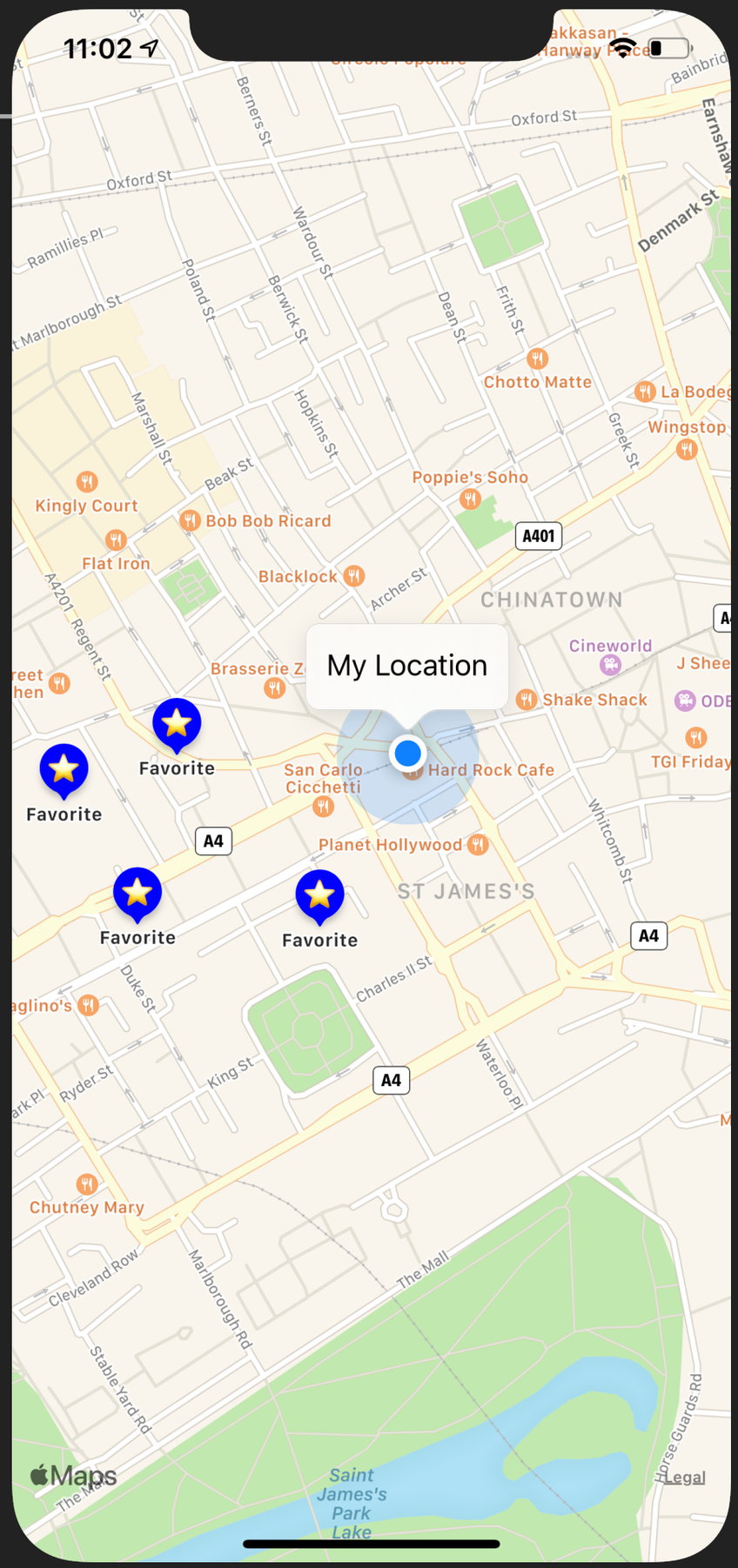
- An annotation view's `collisionMode` determines when a collision occurs between two annotation views
- Could be the border of the annotation view or a custom offset



USER'S LOCATION

- To show the user's location on the map and keep the camera centered on it:

```
mapView.showsUserLocation = true
mapView.userTrackingMode = .follow
```

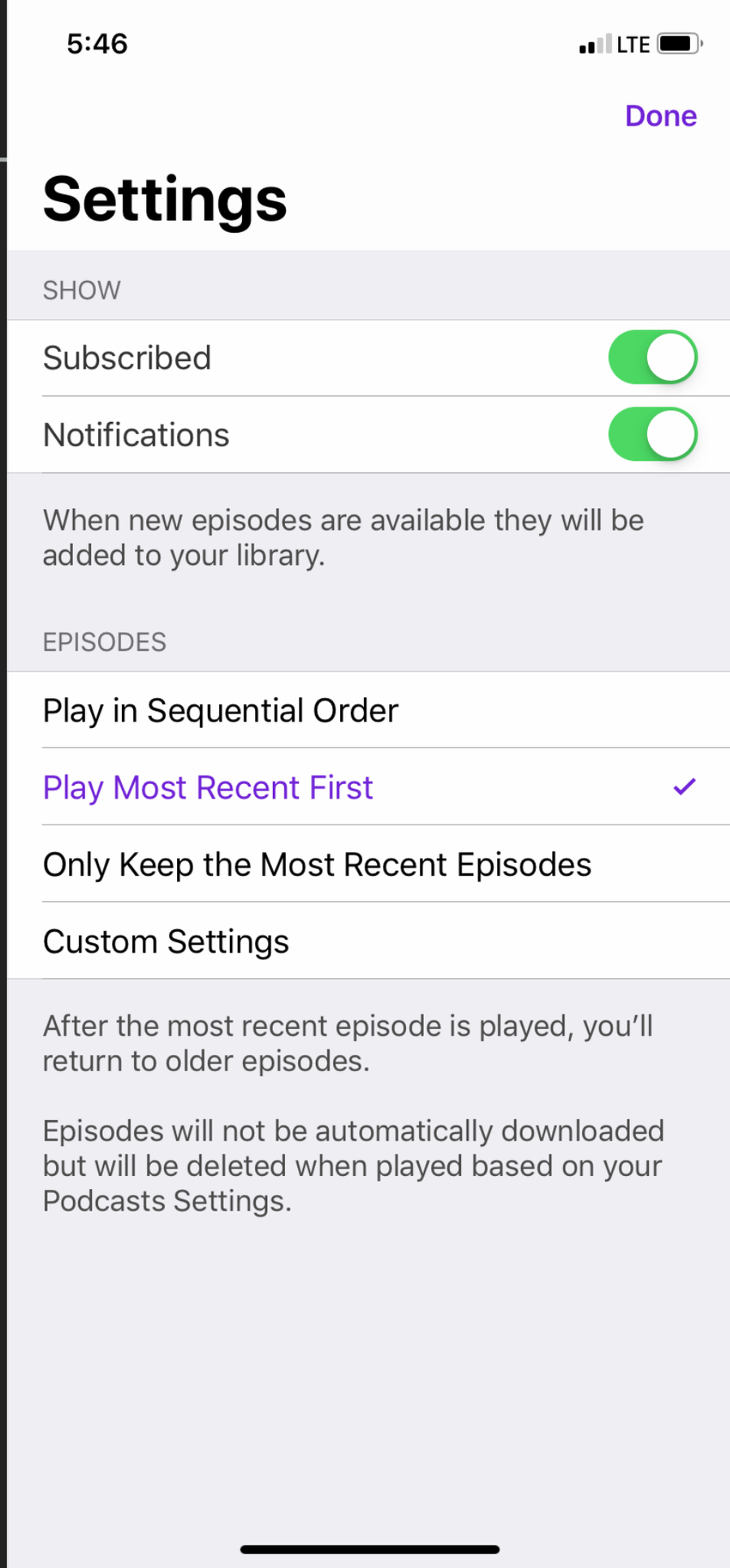


SAVING BASIC TYPES WITH

USER DEFAULTS

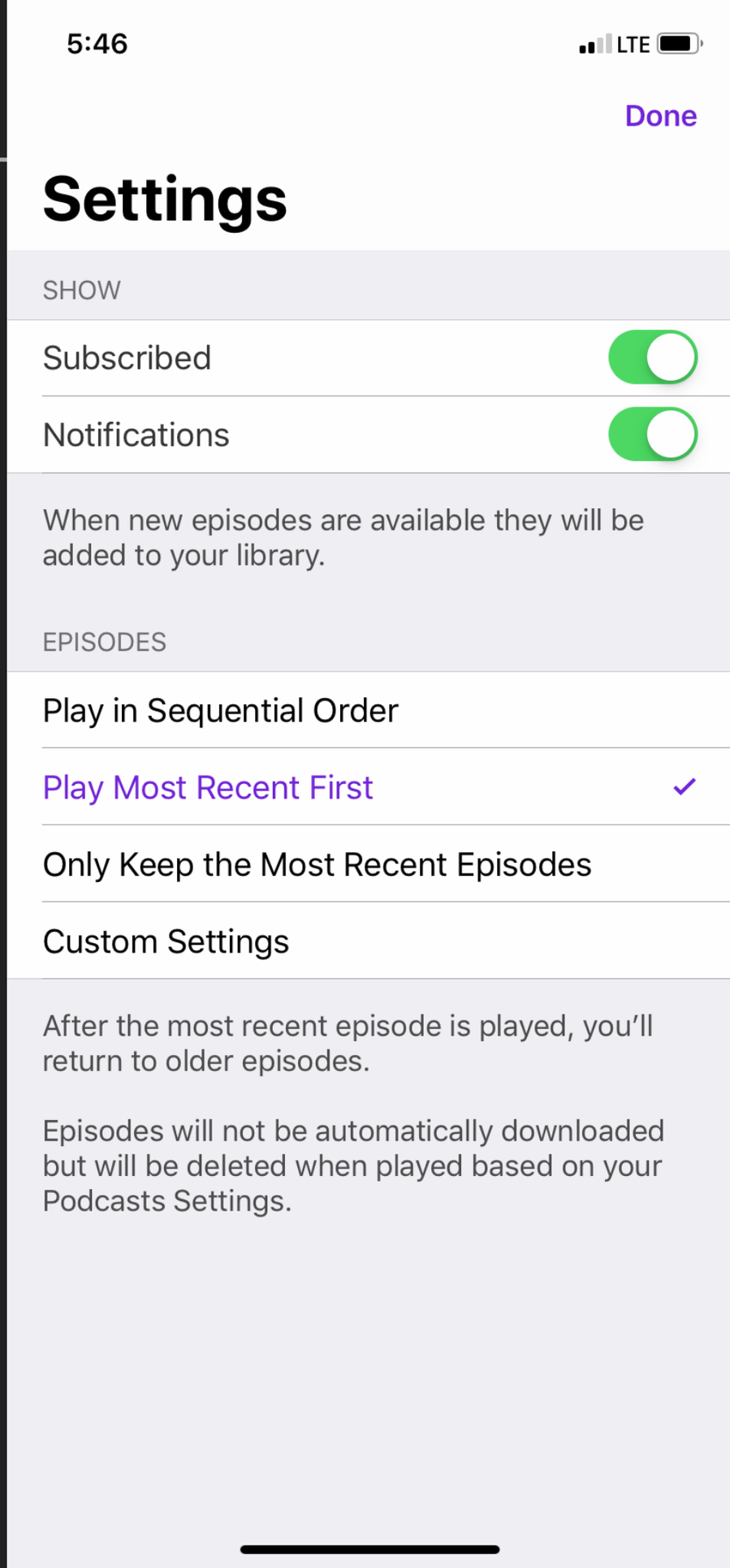
USER DEFAULTS

- Intended to store small amounts of data
  - Example: order of episodes



USER DEFAULTS

- Typically used to restore an app to the user's preferred default state
- But... you can store any values here



SAVING A VALUE

- Associate a value with a key when saving

```
UserDefaults.standard.set(nameTextField.text, forKey: "name")
```

RETRIEVING A VALUE

- Retrieve the value using the same key

```
nameTextField.text = UserDefaults.standard.string(forKey: "name")
```

REMOVING A VALUE

- Remove the value for a key

```
UserDefaults.standard.removeObject(forKey: "name")
```

TYPES THAT CAN BE SAVED

- `NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, `NSDictionary`
    - All of these types have toll-free bridging to their Swift counterparts

SETTING DEFAULT VALUES

Setting Default Values

```
func set(Any?, forKey: String)
```
Sets the value of the specified default key.

```
func set(Float, forKey: String)
```
Sets the value of the specified default key to the specified float value.

```
func set(Double, forKey: String)
```
Sets the value of the specified default key to the double value.

```
func set(Int, forKey: String)
```
Sets the value of the specified default key to the specified integer value.

```
func set(Bool, forKey: String)
```
Sets the value of the specified default key to the specified Boolean value.

```
func set(URL?, forKey: String)
```
Sets the value of the specified default key to the specified URL.

GETTING DEFAULT VALUES

Getting Default Values

```
func object(forKey: String) -> Any?
```
Returns the object associated with the specified key.

```
func url(forKey: String) -> URL?
```
Returns the URL associated with the specified key.

```
func array(forKey: String) -> [Any]?
```
Returns the array associated with the specified key.

```
func dictionary(forKey: String) -> [String : Any]?
```
Returns the dictionary object associated with the specified key.

```
func string(forKey: String) -> String?
```
Returns the string associated with the specified key.

```
func stringArray(forKey: String) -> [String]?
```
Returns the array of strings associated with the specified key.

```
func data(forKey: String) -> Data?
```
Returns the data object associated with the specified key.

```
func bool(forKey: String) -> Bool
```

REMOVING DEFAULT VALUES

Removing Defaults

```
func removeObject(forKey: String)
```

Removes the value of the specified default key.

SAVING CUSTOM TYPES WITH

USER DEFAULTS

SAVING CUSTOM TYPES

- What if I want to save an object that is not one of the supported types?
    - Supported types: `NSData`, `NSString`, `NSNumber`, `NSDate`, `NSArray`, `NSDictionary`

SAVING CUSTOM TYPES

- To save an instance of a custom type, convert it into `NSData`
- Conforming to the `NSCoding` protocol makes it easy to convert between your custom type and `NSData`

NSCODING

- `NSCoding` has two methods that you must implement

```
protocol NSCoding {
    init?(coder aDecoder: NSCoder)
    func encode(with aCoder: NSCoder)
}
```

NSOBJECT

- You also need to subclass `NSObject` , which allows the object to be treated like an Objective C type

IMPLEMENTING NSCODING

```
class Book: NSObject, NSCoding {

    let title: String
    let author: String
    let category: BookCategory

    init?(title: String?, author: String?, category: BookCategory?) {•••}

    required convenience init?(coder aDecoder: NSCoder) {
        let title = aDecoder.decodeObject(forKey: "title") as? String
        let author = aDecoder.decodeObject(forKey: "author") as? String
        let categoryRawValue = aDecoder.decodeInteger(forKey: "category")
        let category = BookCategory(rawValue: categoryRawValue)

        self.init(title: title, author: author, category: category)
    }

    func encode(with aCoder: NSCoder) {
        aCoder.encode(title, forKey: "title")
        aCoder.encode(author, forKey: "author")
        aCoder.encode(category.rawValue, forKey: "category")
    }
}
```

NSKEYEDARCHIVER

- Use `NSKeyedArchiver` to convert the object into `Data`

```
do {
    let data = try NSKeyedArchiver.archivedData(withRootObject: books, requiringSecureCoding: false)
    UserDefaults.standard.set(data, forKey: "books")
} catch (let error) {
    print("Error saving books to user defaults: \(error)")
}
```

NSKEYEDUNARCHIVER

- Use `NSKeyedUnarchiver` to unarchive the `Data` back into your custom object

```
var books: [Book]?
do {
    books = try NSKeyedUnarchiver.unarchiveTopLevelObjectWithData(data) as? [Book]
} catch (let error) {
    print("Error fetching books from user defaults: \(error)")
}
```

WHAT IS NSSECURECODING?

- There is a potential security flaw here:

```
class Book: NSObject, NSCoding {

    let title: String
    let author: String
    let category: BookCategory

    init?(title: String?, author: String?, category: BookCategory?) {•••}

    required convenience init?(coder aDecoder: NSCoder) {
        let title = aDecoder.decodeObject(forKey: "title") as? String
        let author = aDecoder.decodeObject(forKey: "author") as? String
        let categoryRawValue = aDecoder.decodeInteger(forKey: "category")
        let category = BookCategory(rawValue: categoryRawValue)

        self.init(title: title, author: author, category: category)
    }

    func encode(with aCoder: NSCoder) {
        aCoder.encode(title, forKey: "title")
        aCoder.encode(author, forKey: "author")
```

WHAT IS NSSECURECODING?

- You decode an object, bringing it into memory, before checking that it's actually the type you expected

```
class Book: NSObject, NSCoding {

    let title: String
    let author: String
    let category: BookCategory

    init?(title: String?, author: String?, category: BookCategory?) {•••}

    required convenience init?(coder aDecoder: NSCoder) {
        let title = aDecoder.decodeObject(forKey: "title") as? String
        let author = aDecoder.decodeObject(forKey: "author") as? String
        let categoryRawValue = aDecoder.decodeInteger(forKey: "category")
        let category = BookCategory(rawValue: categoryRawValue)

        self.init(title: title, author: author, category: category)
    }

    func encode(with aCoder: NSCoder) {
        aCoder.encode(title, forKey: "title")
        aCoder.encode(author, forKey: "author")
```

WHAT IS NSSECURECODING?

- Safer alternative:

```
class Book: NSObject, NSSecureCoding {

    static var supportsSecureCoding: Bool = true

    let title: String
    let author: String
    let category: BookCategory

    init?(title: String?, author: String?, category: BookCategory?) {•••}

    required convenience init?(coder aDecoder: NSCoder) {
        let title = aDecoder.decodeObject(of: NSString.self, forKey: "title")
        let author = aDecoder.decodeObject(of: NSString.self, forKey: "author")
        let categoryRawValue = aDecoder.decodeInteger(forKey: "category")
        let category = BookCategory(rawValue: categoryRawValue)

        self.init(title: title as String?, author: author as String?, category: category)
    }
```

TYPE CHECK OCCURS BEFORE THE OBJECT IS LOADED INTO MEMORY

WHAT IS NSSECURECODING?

- Safer alternative:

CONFORM TO NSSECURECODING

```
class Book: NSObject, NSSecureCoding {

    static var supportsSecureCoding: Bool = true

    let title: String
    let author: String
    let category: BookCategory

    init?(title: String?, author: String?, category: BookCategory?) {•••}

    required convenience init?(coder aDecoder: NSCoder) {
        let title = aDecoder.decodeObject(of: NSString.self, forKey: "title")
        let author = aDecoder.decodeObject(of: NSString.self, forKey: "author")
        let categoryRawValue = aDecoder.decodeInteger(forKey: "category")
        let category = BookCategory(rawValue: categoryRawValue)

        self.init(title: title as String?, author: author as String?, category: category)
    }
```

SET SUPPORTSSECURECODING TO TRUE

WHAT IS NSSECURECODING?

- Safer alternative:

SET REQUIRINGSECURECODING TO TRUE

```
do {
    let data = try NSKeyedArchiver.archivedData(withRootObject: books, requiringSecureCoding: true)
    UserDefaults.standard.set(data, forKey: "books")
} catch (let error) {
    print("Error saving books to user defaults: \(error)")
}
```

REACHING USERS THROUGH

NOTIFICATIONS

NOTIFICATIONS

- You've worked hard to build an app. Now how do you get people to use it?

USER-FACING NOTIFICATIONS

- Notifications let you reach users even when your app isn't running, so that you can:
  - Tell users when new content is available
  - Remind users about events and to-do items
  - Ask users to take a quick action

SILENT NOTIFICATIONS

- Notifications also let you wake up your app so that it can perform a background refresh
  - These are called silent notifications
  - This allows your app to show the latest content as soon as it’s launched
  - Users have no way of knowing about silent notifications (no badge, alert or sound)

SENDING NOTIFICATIONS

- There are two ways to send notifications
  - Local notifications are sent from the device
  - Push notifications are sent from a remote server

LOCAL NOTIFICATIONS

- Local notifications are created by the app
- Possible triggers are:
  - Time-based (e.g., 10 minutes from now)
  - Calendar-based (e.g., every morning at 9am)
  - Location-based (e.g., when user leaves their house)

PUSH NOTIFICATIONS

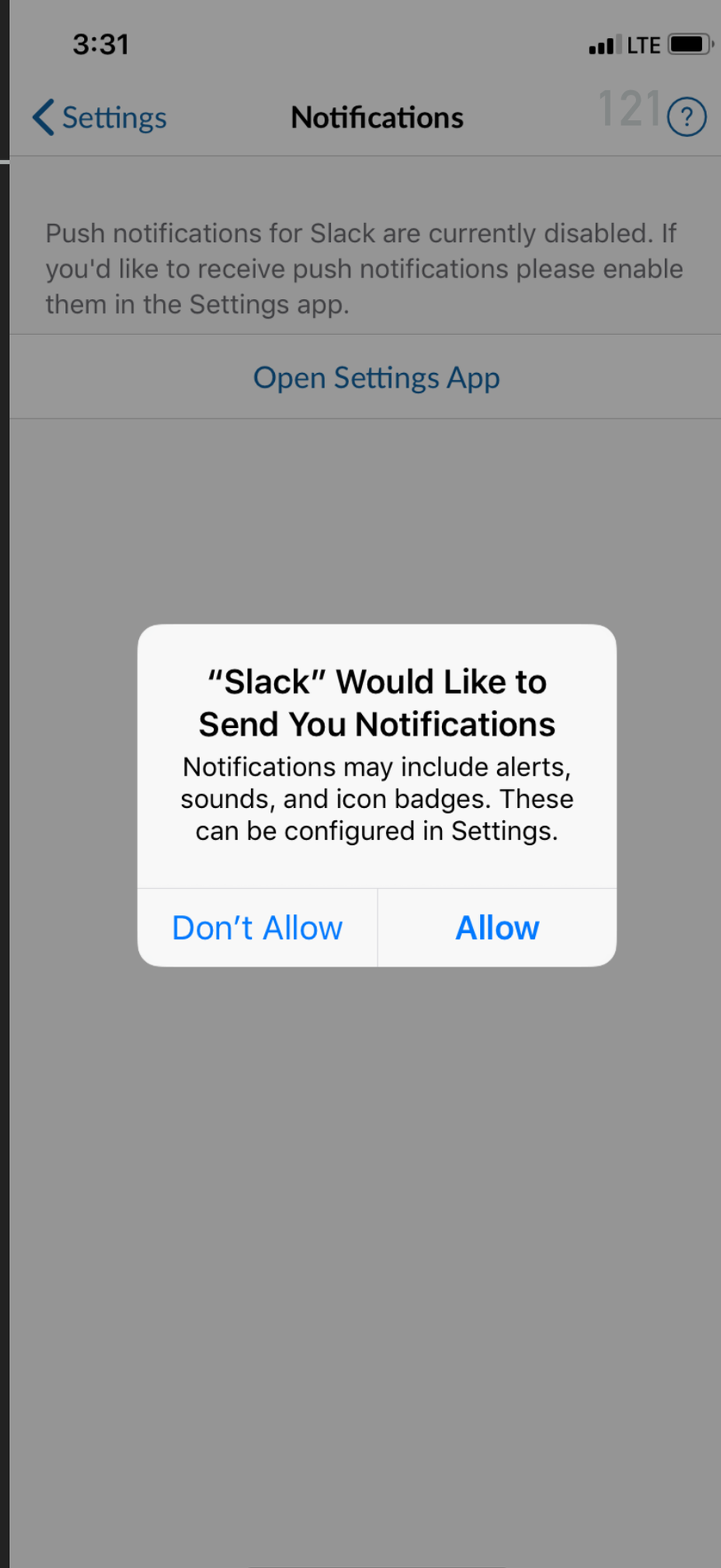
- Push notifications originate on a remote server
    - They must go through Apple Push Notification service (APNs)
- APNs sends the notification to the user's device

NOTIFICATION

BEST PRACTICES

BEST PRACTICES

- Tell the user why your app wants to send them notifications
  - You get one chance to show the system prompt!



BEST PRACTICES

- Ask the user for permission right after they take an action that could be enhanced by notifications
    - E.g., setting a reminder, accepting a friend request

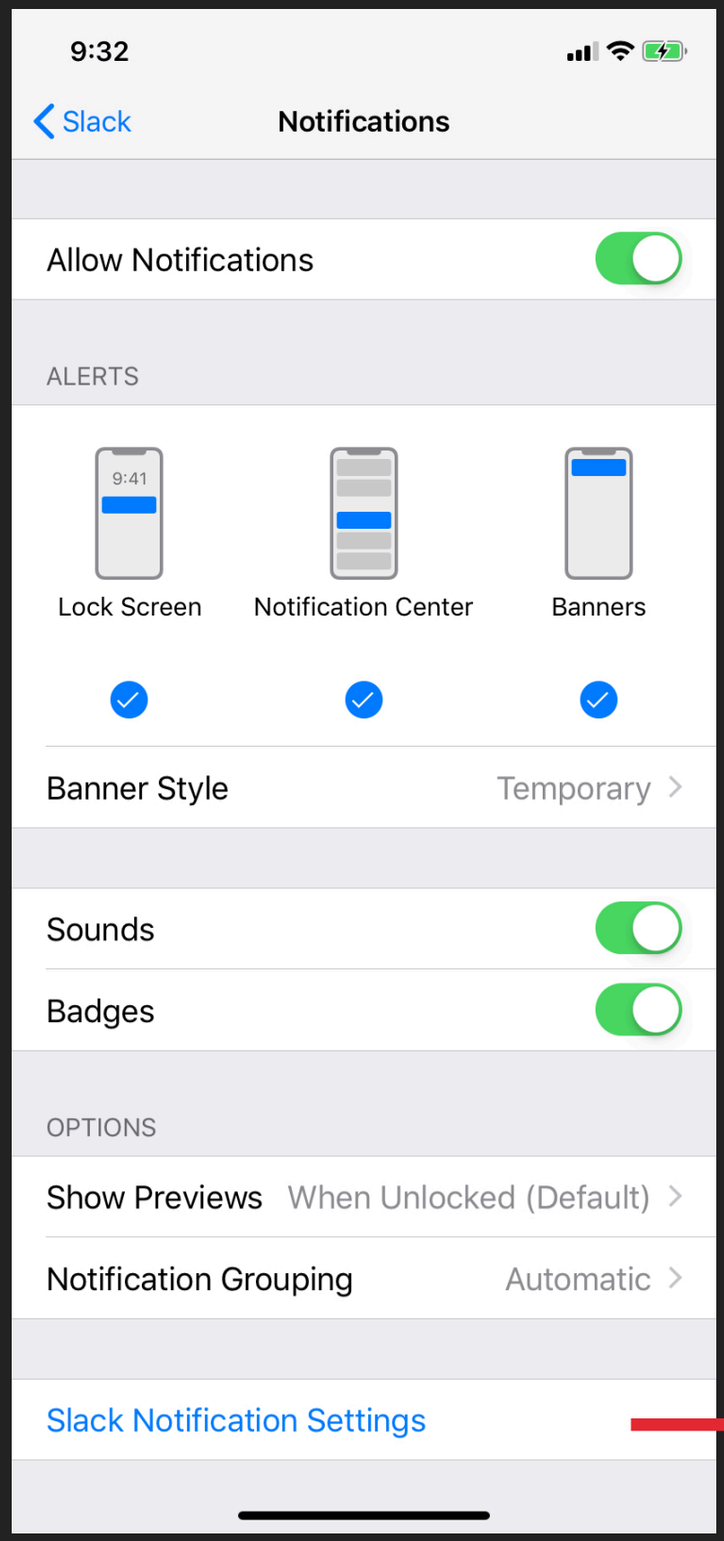
BEST PRACTICES

- Avoid overwhelming your users:
  - Send relevant notifications, not spam
  - Group notifications together
  - Provide options for fine-tuning notification settings
- Remember that users will disable notifications if you bug them :)

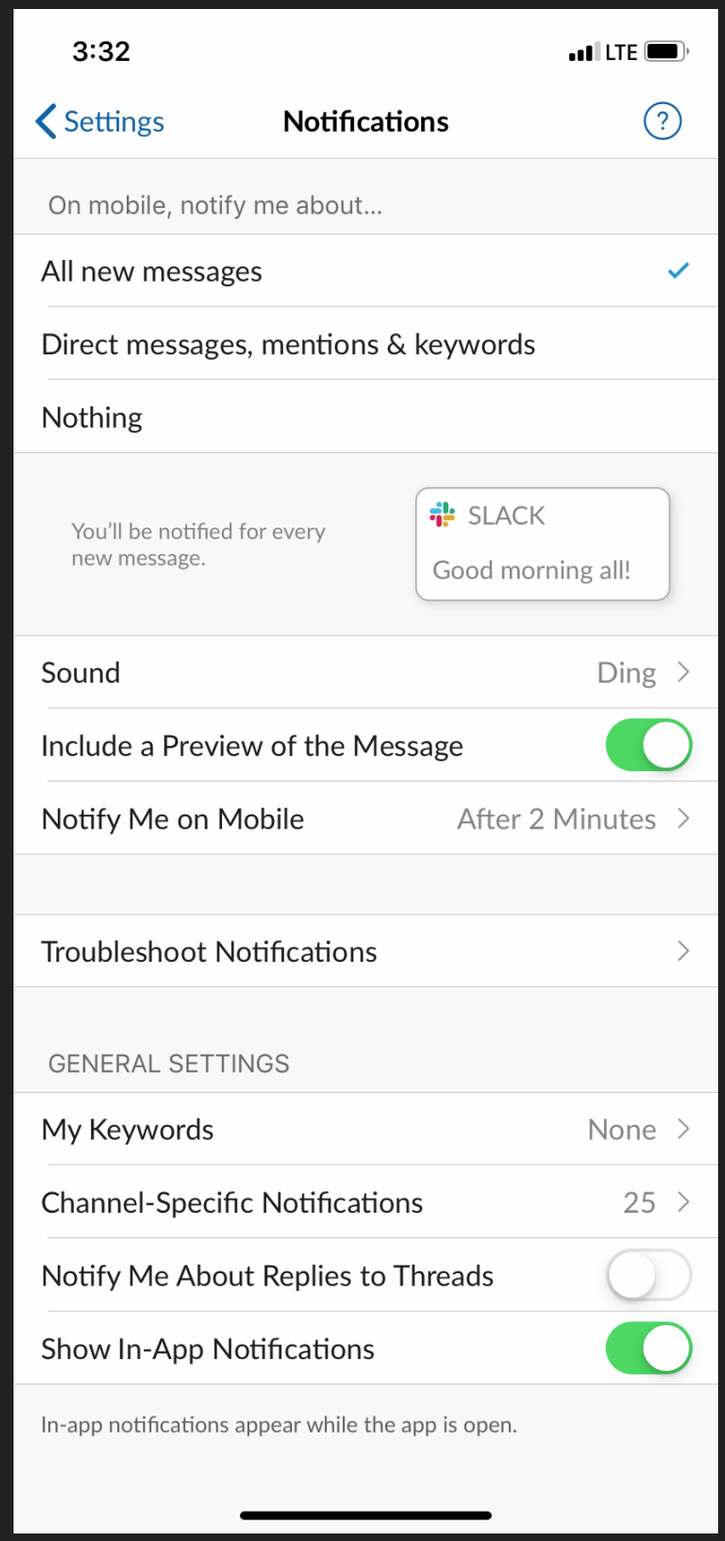
BEST PRACTICES

- Example: Slack

Settings App

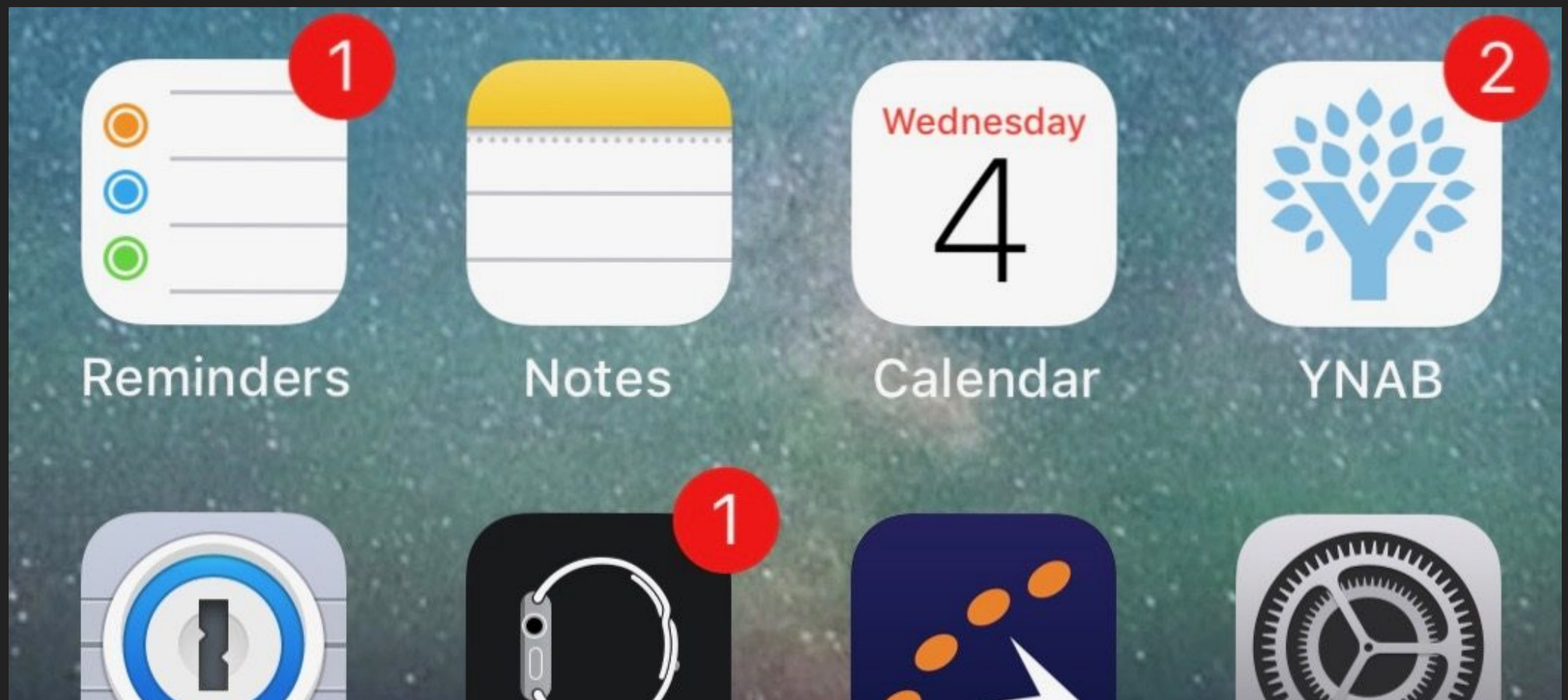


Slack App



BEST PRACTICES

- Give users an easy and obvious way to clear badges



ASKING FOR NOTIFICATION

AUTHORIZATION

AUTHORIZATION

- You must get the user's permission before sending them notifications*

*Usually! We'll discuss an exception.

REQUESTING AUTHORIZATION

- `UNUserNotificationCenter` manages all notification-related behavior
  - Use it's `current()` method to get a reference to the shared object
- Not to be confused with `NotificationCenter`

```
UNUserNotificationCenter.current()
```

REQUESTING AUTHORIZATION

- Call the `requestAuthorization` method to ask for the user's permission to send notification
  - First parameter is a list of options
  - Second parameter is a completion block

Summary

Requests authorization to interact with the user when local and remote notifications are delivered to the user's device.

Declaration

```
func requestAuthorization(options: UNAuthorizationOptions = [],
completionHandler: @escaping (Bool, Error?) -> Void)
```

Discussion

If your app's local or remote notifications involve user interactions, you must request authorization for the system to perfom those interactions on your app's behalf.

AUTHORIZATION OPTIONS

- The notification options you can request are:
  - badge
  - sound
  - alert
  - carPlay
  - criticalAlert
  - provisional

CRITICAL ALERTS

- Normally, notifications are not delivered when the device is in Do Not Disturb mode
- Also, notification sounds are not played when a device is muted

CRITICAL ALERTS

- You can request approval from Apple to send notifications anyway if you have a special use-case, such as:
  - Medical and health alerts
  - Home security alerts
  - Public safety alerts

PROVISIONAL NOTIFICATIONS

- Send (some) notifications before user explicitly grants permission
    - This allows users to see what kinds of notifications your app will send
    - They will be asked if they want to keep receiving notifications in the notification itself

REQUESTING AUTHORIZATION

- The completion block for `requestAuthorization` tells you whether or not the user granted permission (or an error occurred)

```
UNUserNotificationCenter.current().requestAuthorization(options: [.badge, .alert]) { (isGranted, error) in
    guard isGranted && error == nil else { return }
    print("user granted authorization")
}
```

REQUESTING AUTHORIZATION

- Note: The completion block may be called on a background queue

```
UNUserNotificationCenter.current().requestAuthorization(options: [.badge, .alert]) { (isGranted, error) in
    guard isGranted && error == nil else { return }
    print("user granted authorization")
}
```

CHECKING AUTHORIZATION STATUS

- Users can change notification settings for your app later
- To check the current settings, use the `getNotificationSettings` method

```
UNUserNotificationCenter.current().getNotificationSettings { settings in
    if settings.authorizationStatus == .authorized {
        print("notifications authorized")
    }

    if settings.badgeSetting == .enabled {
        print("badges enabled")
    }

    if settings.alertStyle == .banner {
        print("banner style alerts enabled")
    }
}
```

CHECKING AUTHORIZATION STATUS

- This allows you to check whether a user has authorized notifications and how they are configured (sounds, badges, etc.)

```
UNUserNotificationCenter.current().getNotificationSettings { settings in
    if settings.authorizationStatus == .authorized {
        print("notifications authorized")
    }

    if settings.badgeSetting == .enabled {
        print("badges enabled")
    }

    if settings.alertStyle == .banner {
        print("banner style alerts enabled")
    }
}
```

SENDING

LOCAL NOTIFICATIONS

LOCAL NOTIFICATIONS

- To send a local notification, first create a `UNNotificationRequest` with:
  - Identifier
  - Content
  - Trigger

IDENTIFIER

- You can use the `UNNotificationRequest`'s identifier to remove a notification later
    - Example: User deletes a recurring reminder

IDENTIFIER

- What happens if two notifications have the same identifier?
    - The system removes the older one
    - This helps prevent duplicates

CONTENT

- Create a `UNNotificationContent` object to specify how the notification should be displayed to the user
- This includes properties like:
  - Title, subtitle and body
  - Badge
  - Sound
  - Attachments (audio, images, videos)

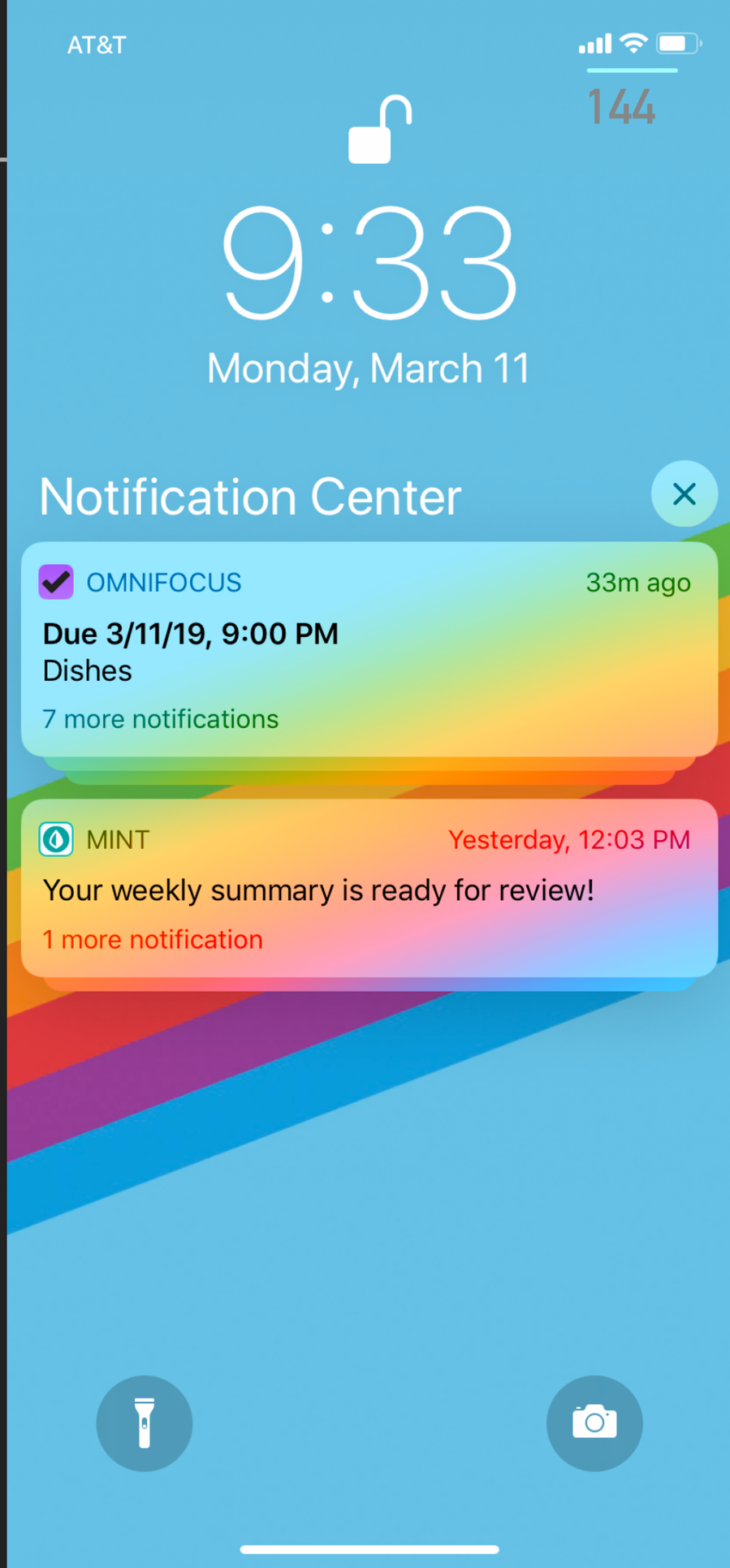
CONTENT

- Create a `UNNotificationContent` object to specify how the notification should be displayed to the user

```
let content = UNMutableNotificationContent()

content.title = "Today at 1pm"
content.subtitle = "Lunch with Phil"
content.body = "Meet at the restaurant"
content.sound = UNNotificationSound(named: UNNotificationSoundName("ding"))
content.badge = 12
```

CONTENT

- You can also set the content's `threadIdentifier`
  - Notifications are grouped by thread identifier on the home screen



TRIGGER

- Create a `UNNotificationTrigger` to specify when the notification should be sent
- Choose from:
  - Time interval trigger
  - Calendar trigger
  - Location trigger

TIME INTERVAL TRIGGER

- Specify a time interval (in seconds)
- Specify whether or not it should repeat

```
// Send notification in 10 minutes
let trigger = UNTimeIntervalNotificationTrigger(timeInterval: (10*60), repeats: false)
```

CALENDAR TRIGGER

- Create a `DateComponents` object to specify the exact day and/or time a notification should fire
- Specify whether or not it should repeat

```
var dateComponents = DateComponents()
dateComponents.hour = 8
dateComponents.minute = 30
let trigger = UNCalendarNotificationTrigger(dateMatching: dateComponents, repeats: true)
```

LOCATION TRIGGER

- Create a `CLRegion` object to specify the region
- Notify on entry, exit or both
- Specify whether or not it should repeat

```
let center = CLLocationCoordinate2D(latitude: 37.335400, longitude: -122.009201)
let region = CLCircularRegion(center: center, radius: 2000.0, identifier: "Headquarters")
region.notifyOnEntry = true
region.notifyOnExit = false
let trigger = UNLocationNotificationTrigger(region: region, repeats: false)
```

LOCATION TRIGGER

- For location triggers, your app also needs the user's permission to access their location

```
let center = CLLocationCoordinate2D(latitude: 37.335400, longitude: -122.009201)
let region = CLCircularRegion(center: center, radius: 2000.0, identifier: "Headquarters")
region.notifyOnEntry = true
region.notifyOnExit = false
let trigger = UNLocationNotificationTrigger(region: region, repeats: false)
```

SCHEDULING THE NOTIFICATION

- Once you have your identifier, content and trigger, call `add` to schedule the notification
- If an error occurs, access it through the completion block

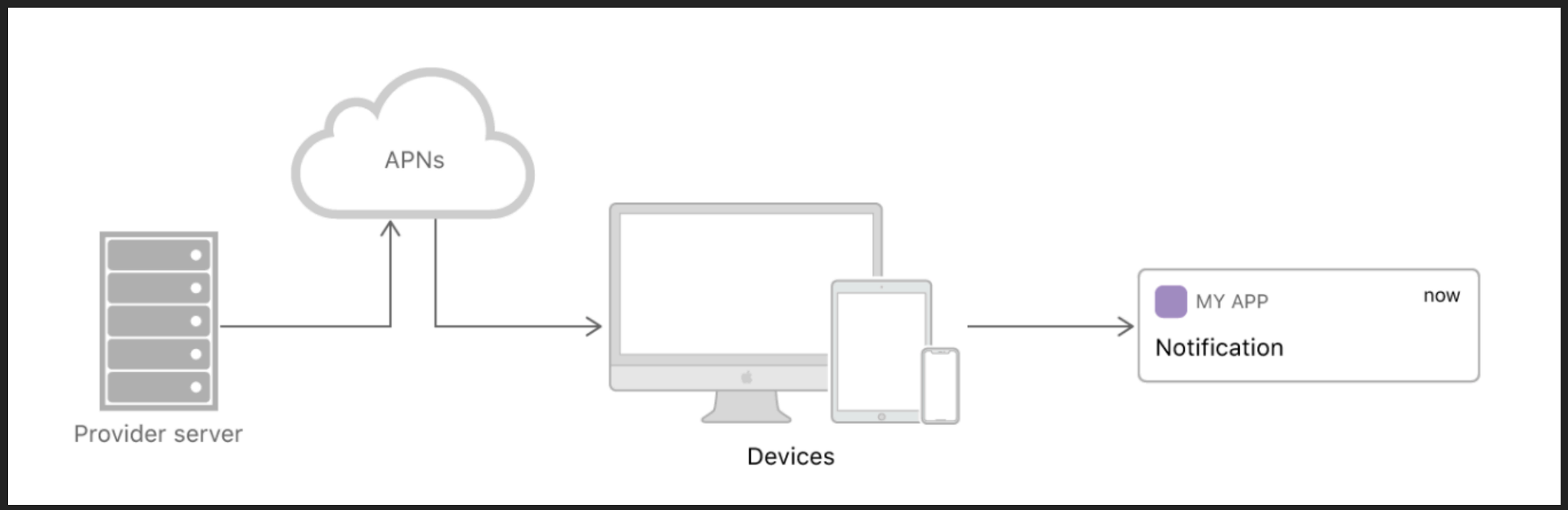
```
let request = UNNotificationRequest(identifier: "my-notification", content: content, trigger: trigger)

UNUserNotificationCenter.current().add(request) { error in
    if let error = error {
        print(error.localizedDescription)
    }
}
```

SENDING

PUSH NOTIFICATIONS

PUSH NOTIFICATIONS

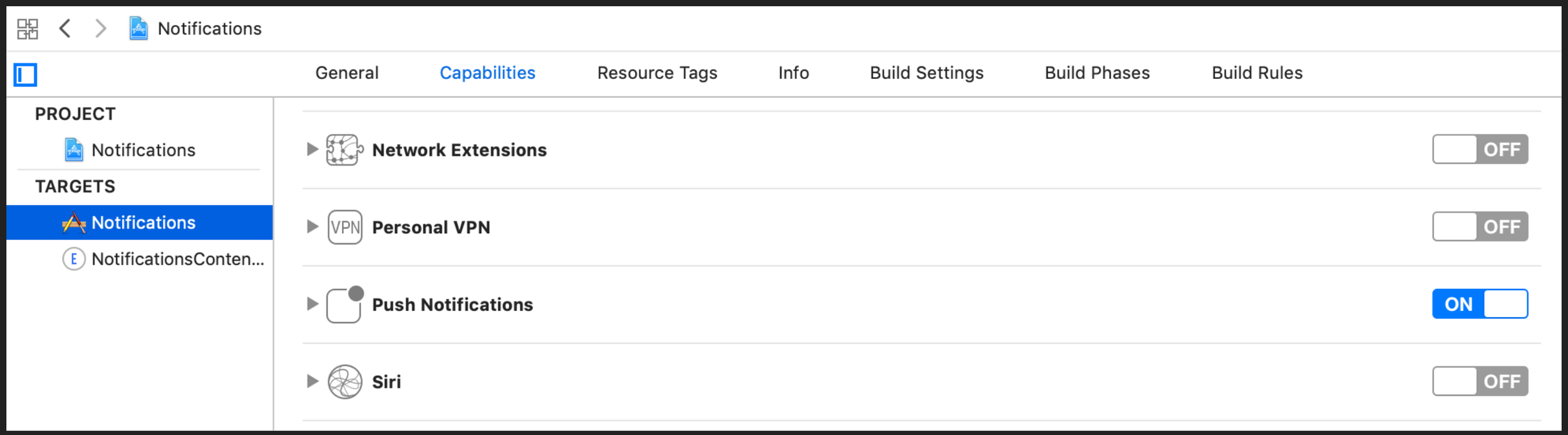
- Push notifications are sent from a remote server, rather than being scheduled on the device



https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server

PUSH NOTIFICATIONS

- To enable push notifications, you must turn on the “Push Notifications” capability
    - Note: This requires a paid developer account



PUSH NOTIFICATIONS

- Step 1: Register for a device token
    - This tells APNs that you want to send push notifications to this device
    - Application launch is a good time to do this

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
    [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    UIApplication.shared.registerForRemoteNotifications()
    return true
}
```

PUSH NOTIFICATIONS

- Step 1: Register for a device token
    - If registration is successful, you'll receive token through an AppDelegate method
    - Send this token to your remote server

```
func application(_ application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {
    print("received device token: \(deviceToken)")
}

func application(_ application: UIApplication, didFailToRegisterForRemoteNotificationsWithError error: Error) {
    print("Did not register for remote notifications: \(error.localizedDescription)")
}
```

PUSH NOTIFICATIONS

- Step 1: Register for a device token
    - If registration fails (because device is offline, APNs is unreachable, etc.), you'll receive an error instead

```
func application(_ application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {
    print("received device token: \(deviceToken)")
}

func application(_ application: UIApplication, didFailToRegisterForRemoteNotificationsWithError error: Error) {
    print("Did not register for remote notifications: \(error.localizedDescription)")
}
```

PUSH NOTIFICATIONS

- Step 1: Register for a device token
    - The device token can change (for example, when user restores the device from backup)
- It’s a good idea to get a new token regularly

PUSH NOTIFICATIONS

- Step 2: Send notification to APNs as JSON
  - Apple-defined keys go in the `aps` dictionary

```
{
    "aps": {
        "alert": {
            "title": "New Joke of the Day"
            "body": "Why did the chicken cross the road?"
        },
        "badge": 12,
        "sound": "default",
    }
    "joke-id": "12345"
}
```

PUSH NOTIFICATIONS

- Step 2: Send notification to APNs as JSON
  - Custom keys go outside the `aps` dictionary

```
{
    "aps": {
        "alert": {
            "title": "New Joke of the Day"
            "body": "Why did the chicken cross the road?"
        },
        "badge": 12,
        "sound": "default",
    }
    "joke-id": "12345"
}
```

PUSH NOTIFICATIONS

- Step 3: Modify content of notification (if needed)
    - Set `mutable-content` to 1 if you need to modify content on the device before the user receives the notification

```
{
    "aps": {
        "alert": {
            "title": "New Sensitive Info"
        },
        "mutable-content": 1
    }
    "encrypted-data": "some encrypted data"
}
```

PUSH NOTIFICATIONS

- Step 3: Modify content of notification if needed
    - Instead of being delivered to the user directly, the notification will go to your Notification Service Extension first

```
{
    "aps": {
        "alert": {
            "title": "New Sensitive Info"
        },
        "mutable-content": 1
    }
    "encrypted-data": "some encrypted data"
}
```

PUSH NOTIFICATIONS

- Step 3: Modify content of notification if needed
    - This gives you a chance to decrypt sensitive data, download images or perform any other modifications

```
{
    "aps": {
        "alert": {
            "title": "New Sensitive Info"
        },
        "mutable-content": 1
    }
    "encrypted-data": "some encrypted data"
}
```

PUSH NOTIFICATIONS

- Step 3: Modify content of notification if needed
  - Warning: You only have 30 seconds! ⏰
  - This *should* actually be more than enough time

PUSH NOTIFICATIONS

- Sending a silent notification
  - Set `content-available` to 1
- Instead of notifying the user with a badge, alert or sound, the notification will wake up your app

```
{
    "aps": {
        "content-available": 1
    }
    "custom-key": "info for background refresh"
}
```

PUSH NOTIFICATIONS

- Sending a silent notification
    - AppDelegate's `didReceiveRemoteNotification` method will be called
    - You have 30 seconds to complete any tasks and call the completion handler

```
func application(_ application: UIApplication, didReceiveRemoteNotification userInfo: [AnyHashable : Any],
    fetchCompletionHandler completionHandler: @escaping (UIBackgroundFetchResult) -> Void) {
    // perform background refresh and call completion handler
}
```

PUSH NOTIFICATIONS

- Sending a silent notification
  - Apple does not guarantee delivery of silent notifications
  - Delivery may be throttled if you send too many (more than 2-3 per hour)